INTERNATIONAL
HELLENIC
UNIVERSITY

Prediction of student performance using dynamic machine learning models

Georgios Chrysogonidis

SID: **3308180002**

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

JANUARY 2021

THESSALONIKI – GREECE

Prediction of student performance using dynamic machine learning models

Georgios Chrysogonidis

SID: **3308180002**

Supervisor: Prof. Konstantinos Diamantaras

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

JANUARY 2021

THESSALONIKI – GREECE

Abstract

This dissertation was written as a part of the MSc in Data Science at the International Hellenic University.

Student performance prediction is closely related to Knowledge Tracing (KT). It is a well-defined problem that the academic community has tried to address and offered competitive results. Over the last two decades, several approaches have been implemented that have contributed to improving existing methods, tackling the problem with different model architecture while experimenting in different datasets and formats. Deep learning-based methods and cutting edge-methodologies that applied to the educational sector improved the performance of the initial methods in the domain of KT. The ability to track the knowledge state could be beneficial in human learning individually and from a business perspective in the early discovering gaps in the learning process to perform a more personalized user learning experience.

The present thesis implements dynamic machine learning models to determine the state of a student based on their past interactions. As a subclass of Deep Neural Networks (DNNs), different algorithms of Recurrent Neural Networks (RNNs) have participated in the experiments in conjunction with the use of Natural Language Processing (NLP) techniques. During the phase of experiments to construct the best model architecture, another category of Deep Learning models, denoted as Convolutional Neural Networks (CNNs), was utilized composing a new framework which demonstrates interesting results. Finally, the thesis is complemented with the research of existing KT models and compares their results with those produced by the model of this work.

Acknowledgments

I wish to express my deepest gratitude to my supervisor, Professor Konstantinos Diamantaras whose support and guidance lead me to the end of this journey. The weekly meetings consisting of tips, clarification of difficult concepts and freedom to experiment and test different implementations proved to be an integral part of my learning process and helped me to choose the right direction to successfully complete my thesis.

I would like to recognize the invaluable assistance of Ph.D. candidate Marina Delianidi; without her deep knowledge of the subject and insightful suggestions, the completion of this thesis would be harder.

Finally, I appreciate the understanding and encouragement I received from my friends and my family and thank them for the happy distractions to rest my mind outside of my research.

Georgios Chrysogonidis

04/01/2021

Contents

Table of Figures

Table of Tables

1 Introduction

In the last few years, online learning has become one of the most fast-growing industries. The massive open online courses (MOOC) and intelligent tutoring systems (ITSs) offer a large collection of courses from different educational sectors attracting much the interest of the public. Those providers accompany their lectures with tests and milestones for users to practice. This integration in their platforms is a two-way learning benefit process. The ability of online learning platforms to track the students' interactions through their logs produce huge amounts of data and have proven to be valuable in educational data mining and learning analytics. [1] Knowledge Tracing (KT) is vital in every online learning and teaching marketplace as it can contribute mainly to a more personalized student learning experience. [2] This can be achieved by utilizing machine and deep learning methods. The need for powerful models that are capable to accurately predict the knowledge state of a student as they interact with online coursework is known as Knowledge Tracing [3] and is the subject of this thesis.

To acquire a deeper understanding of the KT task, Chapter 2 is dedicated to the explanation of the KT definition, its main contributions in the educational sector and the problem structure. Moreover, in the same chapter, the suitability of deep learning methods and especially the sub-category of recurrent neural networks for addressing the KT problem is elaborated. The theoretical background and some practical implementations of the utilized dynamic machine learning methods are elaborated in Chapter 3 to determine the key differences among them and familiarize the reader. Chapter 4 is going deeper into the literature providing details of the previous work with respect to the existing KT models, describing their capabilities and limitations. The KT models that are elaborated in this chapter are either found extensively in the existing literature or derived from the recent decade without any constraints of the underlying techniques and the dataset's format that these models have been using. The first part of this chapter also includes the thesis' research goals and an overview of the search questions defined to address the student performance prediction task.

In Chapter 5 is presented thoroughly the algorithmic evaluation criteria and the methodology that has been followed to conclude to the best's model architecture in conjunction with the datasets that were utilized in this thesis. At the end of this section are clarified the experiment environment and the libraries that have been used. Chapter 6 involves the results produced by the thesis model. Proceeding to the end of this chapter, a comparison in performance is exhibited among this thesis's model and the state-of-the-art models. With the purpose of evaluating the performance of the implemented model, the comparison with the baselines was based mainly on the dataset's split methodology and the availability of the original code of the respective paper which follows the corresponding format. At the end of this work, the conclusions derived from this thesis research and some recommendations for future contribution format Chapter 7 and Chapter 8, respectively.

Many previous KT models implementing dynamic machine learning algorithms to trace the knowledge state of a student have been criticized for their explainability on mastering concepts on test's questions, their capability to retrieve acquired skills of students and their ability to capture long term dependencies in a sequence of exercises. [3] Although, as can been seen in Chapter 4, these models outperformed the initial KT models [1], [4] and [5]. The need of powerful models that can accurately predict the state of student's knowledge is imperative. Thus, this thesis tries to contribute to the educational sector by providing a new architecture that combines the power of RNNs with the Convolutional Neural Networks and Word Embeddings, exposing the KT task to a new model that offers competitive results.

1.1 Introduction to dynamic machine learning methods

Since the living environment is composed of fast changes, generally in many domains and specifically in the educational sector, the necessity of adaptive and dynamic solutions is on-demand. As mention above, the research direction of the present thesis addresses the students' performance prediction problem by applying dynamic machine learning algorithms. Before the term dynamic is clarified in the categorization of machine learning algorithms, a series of basic definitions will be elaborated in this section.

There are many different definitions of Deep Learning, a class of machine learning, that can be found in the literature. A unified, general purpose and unrestricted from the type of the network, proposed by a group of researchers defining the Deep Learning as "a process not only to learn the relation among two or more variables but also the knowledge that governs the relation as well as the knowledge that makes sense of the relation". [6] The term 'deep' implies the models' big number of representation transformation levels between the inputs and the outputs. [7] These transformation levels, known as layers, can be viewed as a group of units, also known as neurons, formatting an Artificial Neural Network (ANN). ANNs is the core of Deep Learning and it is no other than machine learning tools that are inspired from the biological brain and tried to simulate it. [8] A layer in the network interacts with the other layers in a sense that it receives inputs that transform with activation functions and pass them to the next layer as inputs. A visual representation of the general architecture of an ANN can be seen in Figure 1 [9].

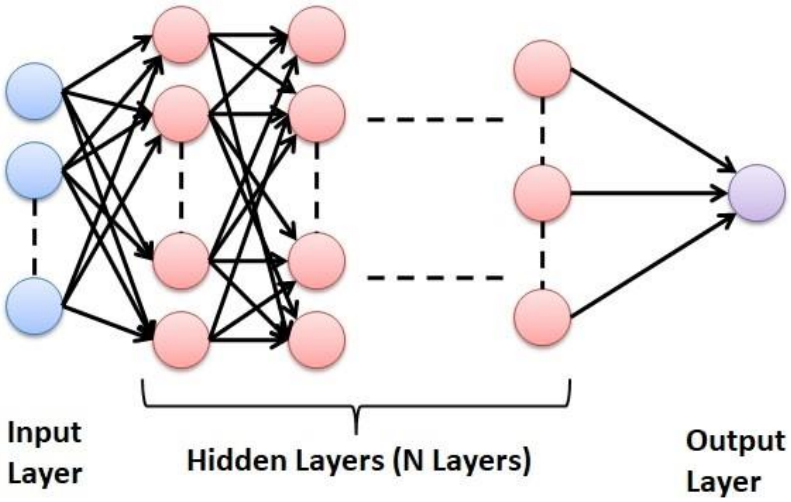

Figure 1: Feed Forward Neural Network architecture. [9]

The family of Artificial Neural Networks contains many types of NNs, each of them is considered task-specific due to their high performance and capabilities. In this thesis, we will focus not only on Recurrent Neural Networks, but on Convolutional Neural Networks too. A sample categorization of those types of ANNs based on their application in real cases can be seen in Table 1.

Table 1: Most appropriate types of ANNs on tasks.

Type	Task
Recurrent Neural Networks	1. Time series analysis 2. Machine translation 3. Image captioning 4. Speech to text
Convolutional Neural Networks	1. Semantic parsing 2. Sentence classification 3. Sentiment analysis 4. Search query retrieval 5. Object and image recognition

Recurrent Neural Networks (RNNs) consist of various dynamic machine learning models that connect neurons over time. [3] The term dynamic refers to the temporal dynamic behavior that is exhibited between the neurons of the network. At each time step *t*, a neuron receives as input the current input of the system and the output of the previous time step *t-1* resulting in a combination of inputs that allows RNNs to access both present and past in order to make decisions. Comparing the previous figure with Figure 2, one can distinguish the existence of feedback between the nodes. These feedbacks, formulate a directed graph which is the general architecture of RNNs. The training of an RNN can be achieved with the Back-Propagation Through Time (BPTT) algorithm. This method helps in minimizing the error between the predicted by the model and the actual output by going backward to update the weights and bias of the network. On the contrary, the Forward Propagation is a procedure to produce the desired outputs by feeding the inputs to the initial layer and propagating them by the hidden layers. These propagations are no other than activations, additions and multiplications that are applied

consecutively the previous hidden layer's values. Forward Propagation is a necessary step in the training of an RNN that precedes the Back Propagation.

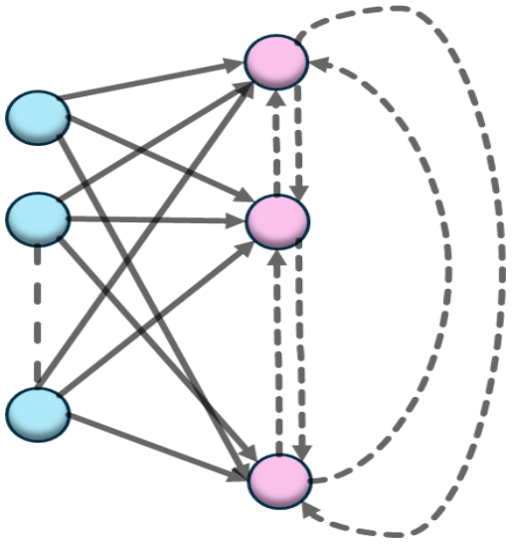

Figure 2: Recurrent Neural Network architecture.

The aforementioned weights' update is determined by the cost function's gradients. The cost function may vary from model to model but the purpose, which is its reduction, remains the same as it increases the performance. Nonetheless, RNNs have a problem that refers to gradients. This RNNs' flaw, named as 'Vanishing Gradient Problem' is elaborated in Chapter 3.

The difference between static machine learning models is that the neurons in each layer has only access to the current input of the system and not their past activation. There are also other dynamic models such as Hidden Markov Models, but the high dimensional representation power of RNNs, especially when it comes to tracing the latent state, made them very popular in the field of education. [3] RNNs include many types such as the Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU) and the Bidirectional LSTM/GRU that participated in the experiments and are explained in detail in this thesis.

2 Knowledge Tracing

This chapter aims to clarify the knowledge tracing problem structure, its importance in education and the suitability of dynamic machine learning models in the KT task.

2.1 Definition and problem structure

The definition of Knowledge Tracing is self-explanatory and appears to be present in all the papers that participated in the literature research, most of them accompanied with a new model architecture such as [2], [3], [4] and [5]. Broadly speaking, KT is to build a model able to track the state of student's knowledge as they answer a series of questions. More specifically, KT is predominantly considered as a supervised sequence learning problem where the goal is to predict the probability that a student will answer correctly the future exercises, given their historical interactions on the current test, identifying the gaps in their learning steps. Mathematically expressed, we seek to find the probability:

$$P(r_{t+1}=1|q_{t+1}, X_t) \quad (1)$$

Where X_t is a set of question-response tuples (q_i, r_i) up to time t, while r_{t+1} is the answer at time $t+1$ and q_{t+1} is the question to be answered at time $t+1$.

The sequence of students' question-answering involves concepts that heavily depend on previous concepts and students must acquire skills, also known as knowledge components (KCs), to correctly answer the next skill's questions. The term KC in the knowledge tracing task is a generic term to define skills, exercises, items etc. [1] Mastering KCs is a non-trivial procedure as it heavily depends on the complexity of human brain and human knowledge [3], each persons' motivation and feeling of competence [10], existing skills, background, the capability to understand, correctly guess, memorize or even forget the underlying notions [4].

2.2 Main contributions of Knowledge Tracing

Knowledge Tracing's contribution to learning activity benefits both students and tutors. With the term 'tutor' we define an online tutoring and learning system that provides courses to students. Tutors can recap exercises to stakeholders that lack the necessary knowledge to move to the next topic of the learning chain. Moreover, tutors can provide to students hints for exercises that involve concepts that were difficult for them to solve. Human experts involving in the Intelligent Tutoring Systems (ITS) could take advantage of the massive datasets produced by the students' engagement and create new teaching materials with richer content concentrated on student's strengths and weaknesses. Even though creating and distributing teaching material is a quite fast process, providing feedback to every student in an online platform individually, referred to their progress on tests and coursework, is a hard and time-consuming work that can assigned to ITS. [11]

On the other hand, students can benefit from tutors' actions by identifying their gaps, focusing more on exercises that are difficult for them to solve and eventually achieving their learning goals. Students have access to high-quality education by minimizing the risk of human mistakes in the feedback process. Finally, they can receive instant feedback, even between each concept's exercises, monitoring step-by-step their performance through the course.

2.3 Suitability of the dynamic machine learning models

Time is an important parameter to the learning equation since students must possess a deep understanding of the undergone concepts to proceed to the next concepts when solving exercises. This results in making the pairs of students' question-answer chronologically ordered leading to the key factor in the selection of the family of models needed to solve the Knowledge Tracing task. Another important parameter is the time distance between two or more dependent concepts in a coursework. To illustrate those dependencies, skills involved in one of the datasets that participated in the experiments are used in the following example. When a student solves exercises that apply the skill "Pythagorean Theorem", they must acquire the ability to solve exercises derived from the skills "Addition and Subtraction", "Exponents" and "Square root". Additionally, since the forgetting factor in skills maintenance is part of human learning and a general char-

acteristic of the human brain, the use of appropriate machine learning models seems necessary. Dynamic machine learning models, and especially LSTM, has the ability to carry the information from input till the prediction to a much later point in time. [12]

3 Artificial Neural Networks

This chapter delves into the theoretical background of the subclasses of Deep Neural Networks that participated in the experiments, focusing on the explanation of the important parts with respect to Knowledge Tracing task.

3.1 Recurrent Neural Networks

A short explanation of RNNs is given in the introduction. Thus, in this section we will focus on the different types of recurrent neural networks, highlighting the differences among them and trying to interpret the mechanisms of these models.

3.1.1 Long Short-Term Memory (LSTM)

LSTM model was first proposed by a Ph.D. student named Sepp Hochreiter and his supervisor Jurgen Schmidhuber in 1997. [13] The creation of this model aimed to solve the "Vanishing Gradient Problem" detected by Sepp Hochreiter in 1991. [14] The gradient descent algorithm tries to detect the cost function's global minimum and update the network's weights by propagating back the gradient error. The derivatives of the activation function tend to become too small when the data contain long sequences. The previous results to even lower gradients due to many multiplications with the same small values. At a single point in time, the gradients may entirely vanish leading to no learning and the training stops. This behavior has a huge impact in the predictions when training a traditional RNN. On the other hand, the gradients may have very large values resulting to "Exploding Gradient Problem" and making the model quite unstable which can lead also to bad performance. The proposed LSTM units provide a different architecture which can propagate the information at a much later point in time solving theoretically the first problem. On the left side of Figure 3, is presented the architecture of an LSTM unit which is composed by a cell state and three gates (input, output and forget gate). [14] Generally, the cell state is responsible for transferring the information from the input to the output and the gates decide which information will be kept or dis-

carded to the current cell state at each timestep. As can be observed in the same figure, each gate contains a sigmoid function that transforms the values from 0 to 1. This is the key component in the decision of "remembering" and "forgetting" data as pass through the cell. If we multiply a number with 0 we get 0 causing these values to be ignored and if we multiply with 1 we get the same value causing them to be kept.

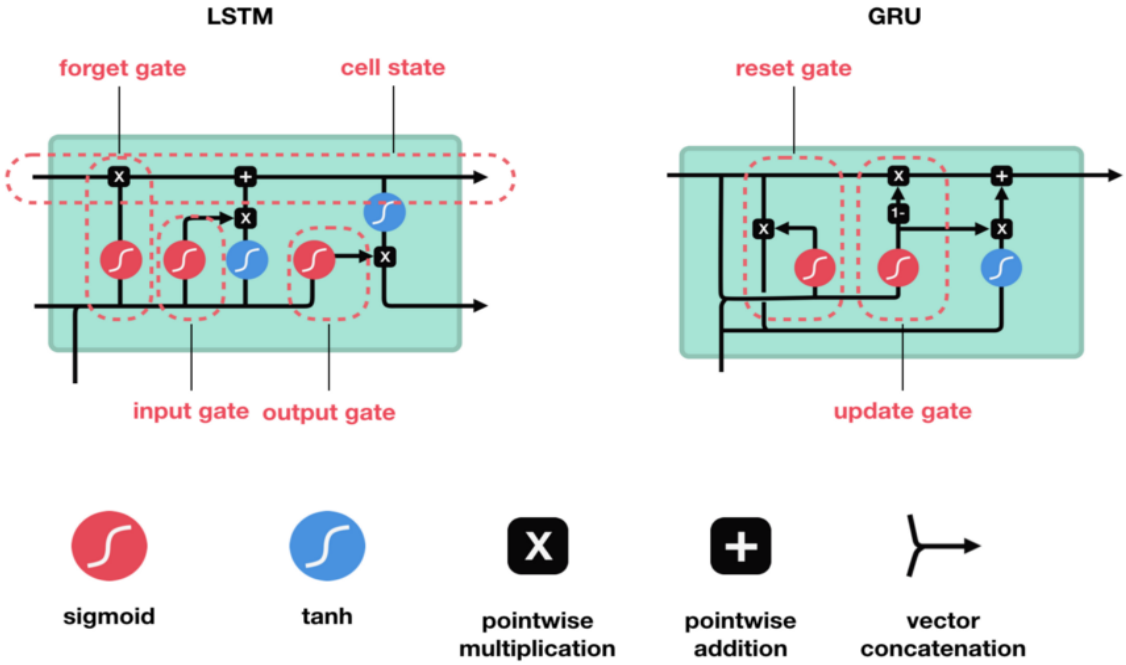

Figure 3: LSTM and GRU units. [15]

More specifically, the forget gate takes as inputs the previous cell's hidden state as well as the current input and pass them into a sigmoid function. The values of this output that are closer to 0 are being "forget" and those close to 1 are being "kept". The input gate takes the same inputs of the forget gate and passes them into a sigmoid and a tanh function separately. The corresponding outputs are then multiplied and added to the current cell state. By this multiplication is decided which values are necessary to be kept to the current's hidden state. Finally, the output gate multiplies the current "tanh-ed" hidden state with the output of the sigmoid function to produce a new hidden state. This gate is responsible for what information will be kept in the next cell's hidden state.

3.1.2 Gated Recurrent Units (GRU)

Gated Recurrent Units (GRU) model is introduced by Lyunghyun Cho in 2014. [16] GRU follows a similar concept with LSTM with a few modifications in its units, removing the separate memory cells. In a GRU unit the hidden cell state of LSTM's is missing

and instead of three gates, there are two, named reset gate and update gate (right side of Figure 3). The reset gate is constructed to deal with the volume of past information being forgotten while the update gate to deal with the volume of past information from previous time steps being passed to the output. The GRU also eliminates the vanishing gradient problem found in vanilla RNN, as it keeps only the relevant information to be passed out to the next time steps. Experiments have shown that GRU has similar performance to LSTM on some task such as in speech recognition [17] and in cases where the dataset is smaller in size and less frequent it can outperform LSTM [18].

3.1.3 Bidirectional LSTM and Bidirectional GRU

Bi-LSTM and Bi-GRU architectures consist of two LSTMs and GRUs respectively. These models process the inputs in a both forward and backward direction. The ability of this deep learning method that can be applied to LSTM, GRU and generally to RNNs proposed by Schuster and Paliwal in 1997. Bidirectional RNN (BRNN) can be considered as an extension of a simple RNN that does not require their input to be in a fixed length. In [19], the authors mention that this method does not limit the training up to a particular time step and claim that it provides better results in many classification and regression problems. This is true especially for entity extraction and translation tasks where the inputs contain contextual word information. [20], [21] The unfolded version of Bi-RNN up to time *i*, is depicted in Figure 4 where the state neurons on positive s_i and negative s_i*'* time direction do not interact with each other.

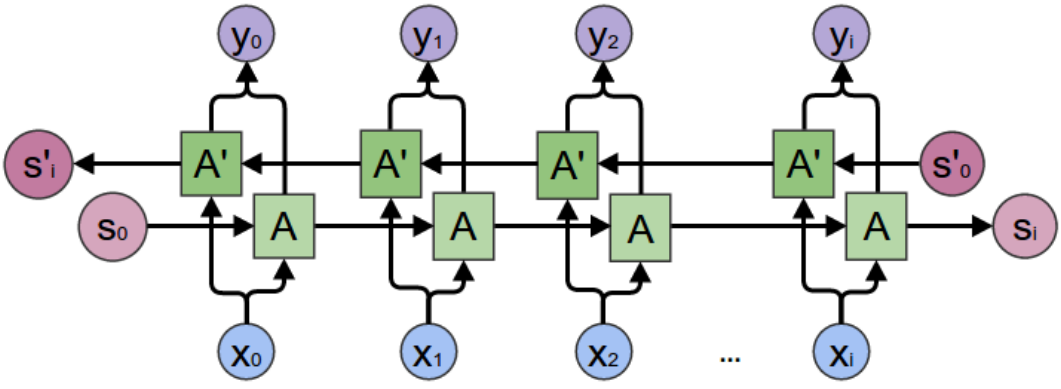

Figure 4: Structure of the bidirectional LSTM (Bi-LSTM) shown unfolded in time for i time steps. [22]

3.2 Convolutional Neural Networks

Another class of ANNs that taken part in the experiments is the Convolutional Neural Networks (CNNs). CNNs are a type of deep neural networks and are regularized cases of Multilayer Networks that are inspired by the structure of the virtual cortex. [23] The latter means that some neurons are constructed by receiving the weighted sums of the previous layer neurons' activations. These neurons come from a specific region known as the receptive fields. The entire area is then covered by many, sometimes, overlapping receptive fields. CNN's architecture may involve several layers such as convolution layers, pooling layers followed usually by fully connected layers. An example of CNN structure can be seen in Figure 5, where, during the training with backpropagation that uses a ReLU as gradient descent optimization function, the kernels and weights are updated based on the value of a loss function. [24] The Convolutional layer extracts features from the data creating a feature map, usually with linear operations that are called convolution, matching the shape of a mask also known as the kernel. The mask is no more than an array of numbers extracted from inputs. The Pooling layer is a subsampling layer that calculates the neurons of the next layer using a mathematical operation that is performed on the neurons of the previous layer's receptive field. The most common pooling layers are Average pooling and Max-pooling and both participated in our experiments. Average pooling calculates the average of the receptive field and Max-pooling takes its maximum.

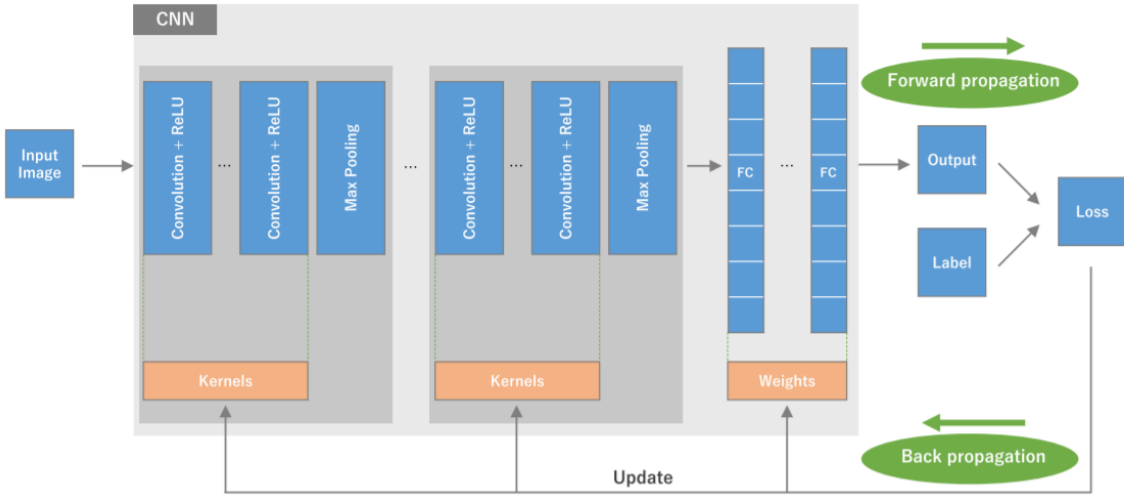

Figure 5: Sample architecture of CNN through training taking as input an image. [24]

3.3 Word embeddings

Word embeddings are an important part of every constructed model architecture in this thesis. Word embeddings are continuous vector space representations of categorical features restricted to user-specified dimensions. The number of dimensions is actually a parameter that could be modified based on the model performance. Initially, word embeddings were created among other reasons to convert categorical variables into numerical since machine learning models cannot process those data types. Neural Networks, theoretically speaking, can approximate any continuous function and the injection of word embeddings in a neural network proved that provides optimization in convergence during training and retains the stability of a network during testing when the values of embeddings are modified. The latter is wrong when a different machine learning method such as Decision Trees are used in conjunction with word embeddings. [25]

Word2Vec is an algorithm created by Google in 2013 aiming to construct word embeddings based on the context, meaning that embeddings of a particular word are generated based on the context of its surrounding words. Word2Vec assumes that words with similar embeddings have similar representations. [26] Word2Vec embeddings are part of the experiments of this work and the techniques to generate those embeddings are elaborated in the next chapters.

FastText can be considered as a descendant of the Word2Vec and was created by Facebook in 2016. This technique is based on the Skip-gram model with a modification in the scoring function to capture the information of the word's structure. The summary of all the n-grams representations that is derived by a word in a training dataset consists the FastText word embeddings of this word. [27] The way that these word embeddings are generated solves the problem of Word2Vec embeddings to construct the continuous vector representation of rare words or even words that are not contained in the corpus. Moreover, in the respective paper, the authors mentioned that FastText offers superior results in many NLP tasks such as machine translation and speeds the training in large collection of words. These word embeddings technique was a latter addition to this thesis so as to provide a way to generate word embeddings from skill names that occurred for the first time in the collection of skill names.

3.4 Attention mechanism

For the purpose of performance improvement of the final model's architecture in this thesis, the attention mechanism was utilized. There are three different kinds of the attention layer, the Additive, the Dot-Product and the Multi-Head attention. The discrimination among them was made based on the way the scores are computed. The attention mechanism introduced to fix the vanishing-gradient problem that RNNs suffer from, focusing on the important parts of each input. RNNs and especially LSTM and GRU, have shown significant results in sequence to sequence (seq2seq) predictions. The original architecture of seq2seq architecture, proposed by Sutskever et al. consists of an encoder and a decoder which are basically two RNNs. The encoder compresses the information of the input into a context vector of fixed length while the decoder, as its name suggests, reconstructs the target sequence. The attention provides information between the input sequence and the decoder output at each time step. In every time step, it is computed an alignment vector containing the score between the input sequence and the decoders output. The context vector is thus a mixing of the alignment vector and the encoder's output. The general architecture of an attention layer can be seen in Figure 6, where the layer receives the query, key and value vector.

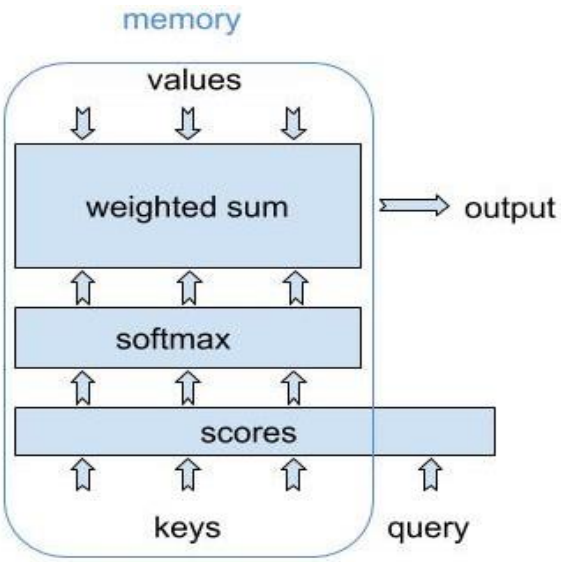

Figure 6: Attention general architecture.

In this thesis we used the Dot-Product and the Additive attention layer. The first is also known as Luong-style attention and was published in 2015 to solve a machine translation task. [28] The Dot-Product attention layer assumes that the dimensions of the queries and the keys are equal, where the queries are the inputs in the attention layer, and

thus by calculating their dot product it tries to extract their similarities. The Additive attention layer, also known as Bahdanau-style attention, was the first proposed attention mechanism in 2014. The difference between this attention layer the previous was the non-linear summary instead of calculation of dot product of queries and keys.

Going deeper into the mathematical interpretation of the attention layer, its output is derived from a series of calculations. The first step is to compute the scores s_i between the query q and the key k_i, for each $i\epsilon(1, n)$.

$$s_i = score(q, ki) \quad (2)$$

Then, we need to normalize the scores with a softmax function.

$$wi = exp(si) / \sum_j exp(s_j) \quad (3)$$

The last step is to calculate the weighted sum of the values v_i and the weights w_i.

$$output = \sum_i w_i v_i \quad (4)$$

4 Related work

This chapter's objective is to present a brief review of the KT literature, explaining the different implementations of the previous attempts over the students' performance prediction problem. However, the hidden purpose of this section is to help the reader better understand the difference between the previous models and this dissertation's model.

4.1 Goal and Research Questions

Several Knowledge Tracing models have been introduced over the last thirty years, proposing different approaches by implementing diverse techniques. We mostly focus on models that have used deep learning-based methods, not only for comparison purposes but as it seems that there is still room for improvements due to the rapid development of powerful models in the current domain.

4.1.1 Goal

The master thesis goal is to construct a new and as simpler as possible model architecture that demonstrates superior performance from the existing KT models. To ensure that this achievement is feasible, a systematic review is required to observe what has been done and what could be performed differently.

4.1.2 Literature Review and Research questions

During the systematic review stage, a series of questions were recorded. These questions were either cited clearly in the relevant papers or were born during the process and we grouped them into *Literature Review Questions (LRQ)* and *Research Questions (RQ)*. The distinction between them has been made based on the origin and intention; the LRQ were based on the literature review with the purpose of exploration and understanding the underlying notions and existing models while the RQ to the experimentation and practical implementation of various techniques to achieve the desired objective. By filtering the most essential, the questions are the following:

LRQ.1. What is Knowledge Tracing and what its contribution is to the educational sector?

LRQ.2. What methods have been used to tackle the knowledge tracing task and what are the differences among them?

RQ.1. Can we contribute to KT by providing a new model with the use of dynamic machine learning methods that demonstrates better performance?

The first question was answered in Chapter 2, the second is presented in the next section and the third is elaborated in the next chapters.

4.2 Related Work

The categorization among existing KT models has been made based on the methods they use and the capabilities they offer. [1] With the general term of capabilities, we refer to the methods that they use, the interpretation of their parameters and their prediction performance. These models are divided into traditional machine learning and deep learning KT models. [4] Nonetheless, graph theory-based models exist in the KT literature approaching the knowledge tracing problem from a different perspective.

4.2.1 Traditional KT models

The first mathematical KT model dates back in 1990s. This was the Bayesian Knowledge Tracing (BKT) which was developed by Corbett and Anderson in 1995. [29] Since this model exists almost in every paper and most of the time is part of the baselines in their experiments, it was considered appropriate to be included a brief explanation in this dissertation. BKT is a structured, shallow, probabilistic graphical model that tried to trace skill acquisition by utilizing a Hidden Markov Model (HMM). In a HMM, the transition between states is governed by the transition probabilities which obey the Markov Propety, meaning that the probability that the system will be in a state at a time *t* depends only on the state that is was before at time *t-1*. Moreover, a HMM is based on the assumption that the states are discrete. Inevitably these properties adopted by BKT making it highly constrained and has been criticized by later models. The authors mention in their paper that in some tests, the model assumes that all students were in the same state of knowledge and the hidden variables are difficult to interpret in contrary to the expectation of the model's performance in the experiments.

Thus, in the paper [30] the authors tried to implement model individualization and in the paper [31] tried to integrate item difficulty in the BKT model.

4.2.2 Deep Learning KT models

This category refers to models derived from Deep Learning methods. The first implemented model was Deep Knowledge Tracing (DKT) introduced by Christopher James Piech in 2015 in his Ph.D.Dissertation. Later the same year, their paper published results that outperformed BKT in the domain of Knowledge Tracing. The proposed model used RNNs and specifically the LSTM model to predict the student's performance based on their past interactions. This model is devoid of human engineering features such as those proposed in the paper "How deep is knowledge tracing', even though using these techniques yielded higher performance in BKT than DKT. The contribution of LSTM in the DKT model lies in the fact that the hidden state of LSTM was considered by them as the latent knowledge state of a student and can carry the information of their past interactions to the output layer. Then, the output layer of the model computes the final prediction which is no more than the probabilities of answering correct a student's Knowledge Component. In Figure 7 is depicted the unfolded architecture of the RNN in the DKT model, where the inputs are converted to a one-hot or compressed representation of a student's question-and-answer while each output is also a vector that contains the probabilities of answering correctly to a particular question.

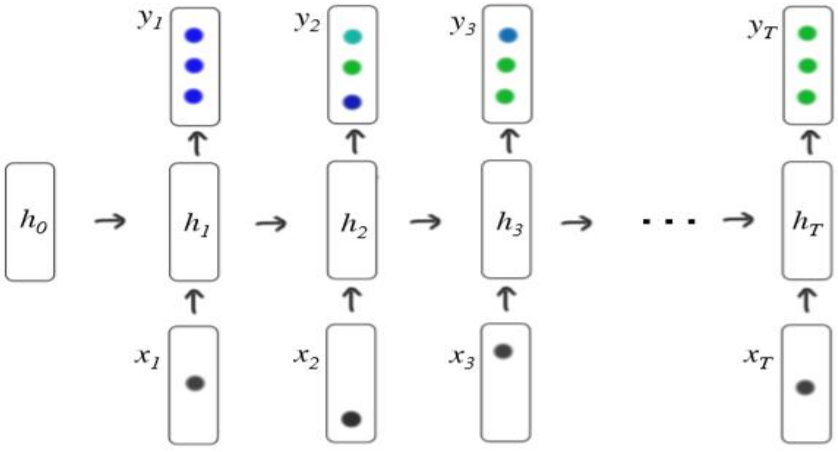

Figure 7: Unfolded architecture of the RNN in DKT. [3]

Two years later four authors from China and Hong Kong proposed a model named Dynamic Key-Value Memory Network (DKVMN) that solves the problems of the previously mentioned model. More specifically, DKVMN tried to capture the relationship

among different concepts that BKT failed so. Moreover, DKVMN solved the lack of DKT to trace each concept state. In the same work, the authors argue that the summary of a student's state of knowledge in a hidden state fails to determine the level of concepts' mastering by a student and even output the skills that a student has learned. The DKVMN model is based on another type of RNN, called Memory Augmented Neural Network (MANN). Basically, in their implementation they extend of standard MANN with a key-value memory as two attention mechanisms trying to predict the knowledge state of a student. DKVMN contains two matrices, one static called *key* and one dynamic which is called *value*. The *key* matrix is responsible for storing the concepts and is immutable and the *value* matrix which is updated when a concept state changes. The latter means that when a student acquires a concept in a test the value in the *value* matrix is updated based on the correlation between exercises and the corresponding concept. The function that computes these matrices in DKVMN alternate from the initial MANN models; the read weight of the *key* matrix is computed by the cosine similarity of an embedding vector that contains the questions and the responses at each timestep and the write weight of *value* matrix is computed by LRUA mechanism.[5] LRUA is technique that used in MANN to write in the recently or the least updated memory location. [32] The DKVMN with these two attention mechanisms outperformed the DKT in all the datasets that participated in their experiments. The differences among MANN and DKVMN structure is represented in Figure 8 and Figure 9, respectively.

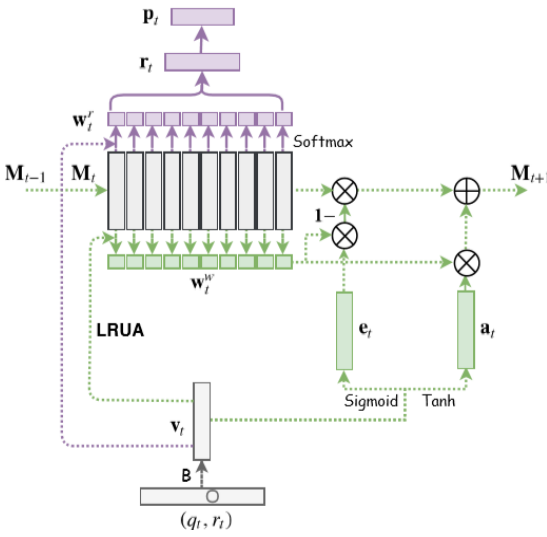

Figure 8: MANN architecture. [5]

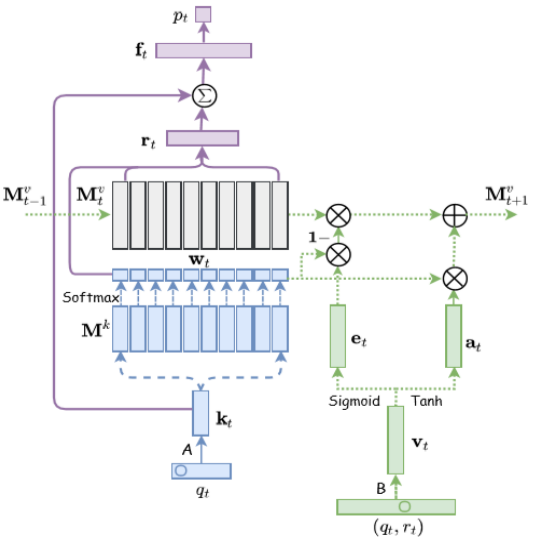

Figure 9: DKVMN architecture. [5]

An extension of DKVMN proposed in 2019 by Chun-Kit Yeung. In his paper named "Deep-IRT: Make Deep Learning Based Knowledge Tracing Explainable Using Item

Response Theory", the author combined the capabilities of DKVMN with the Item Response Theory model (IRT). Generally, Deep-IRT tried to solve the explainability problem of DKVMN. Deep-IRT injects in DKVMN the question's difficulty and each student's ability to correctly answer a question based on their past interactions while retaining its performance in various datasets. Item Response Theory model is a quite old model since it was introduced in 1950s and it is used to calculate the probability that a student defined as *a* will answer a question *j* correctly. This probability is based on the student's ability defined as θ and item's difficulty level β_j. In Figure 10, IRT is mathematically expressed as:

$$P(a) = \sigma(\theta - \beta_j) = \frac{1}{1 + \exp(-(\theta - \beta_j))}$$

where $\sigma(\cdot)$ is the sigmoid function.

Figure 10: Item response function. [1]

The same year, two authors from Australia published a paper with another model named Sequential Key-Value Memory Networks (SKVMN) that tried to overcome the problem of DKVMN to capture long term dependencies in the sequences of exercises and generally in sequential data. The latter has been achieved by them, using a variation of LSTM, called Hop-LSTM in conjunction with a key-value memory. Hop-LSTM's architecture is similar to LSTM's in a way to discover sequential dependencies among exercises, but it skips some LSTM cells to approach previous concepts that are considered relevant. SKVMN also improves the write mechanism of DKVMN by incorporating only the previous knowledge states of a student when it comes to a new exercise and not the current one. [1] The last is referred to cases when a student answers the same question many times and it was solved by SKVMN by summarizing the input vector in the writing procedure. This model exceeded the previous models' performance in their reported experiments and its architecture is depicted in Figure 11.

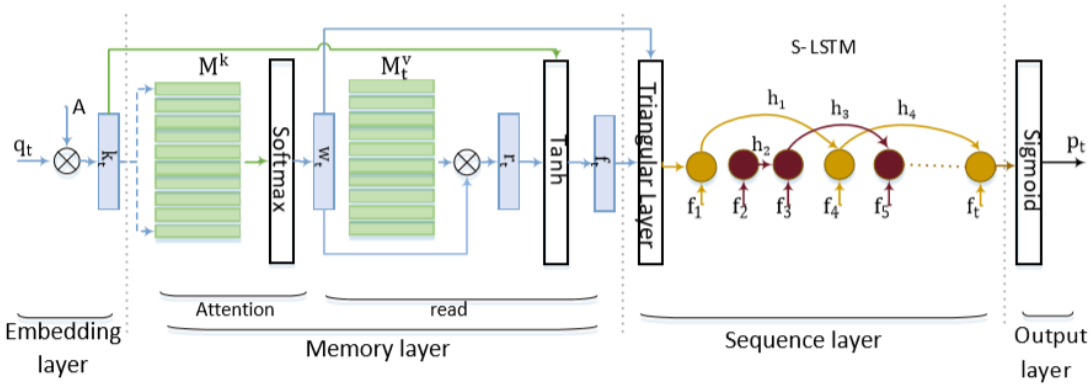

Figure 11: SKVMN architecture. [4]

Exercise-aware Knowledge Tracing with Attention mechanism (EKTA) is another deep learning-based model found in the literature review process which utilizes a Bidirectional LSTM to exploit the content of each exercise and an attention mechanism to gain performance. In the corresponding paper of EKT framework, published in 2019, they proposed another variation by utilizing the Markov property which they named EKTM and demonstrate poorer prediction performance than EKTA. [33] EKTA had better performance improvement compared to BKT, DKT, DKVMN in a single utilized dataset collected from the 'Zhixue' online learning platform. Finally, another newly proposed 'deep' model named Self Attentive Knowledge Tracing (SAKT) published in 2019. SAKT utilizes a self-attention mechanism and is mainly consists of three layers; an embedding layer, a Multi-Head Attention layer and a feed-forward layer. [34] The general architecture of the SAKT can be seen in Figure 12 where the v_i, k_i and q_i are generated from the Embeddings layers shown in Figure 13 and are derived from the student's past interactions x_t and the current's exercise e_{t+1}. Finally, the authors provided results that suggest that their model outperformed the DKT and the DKVMN in their proposed experiments in various well-known KT datasets.

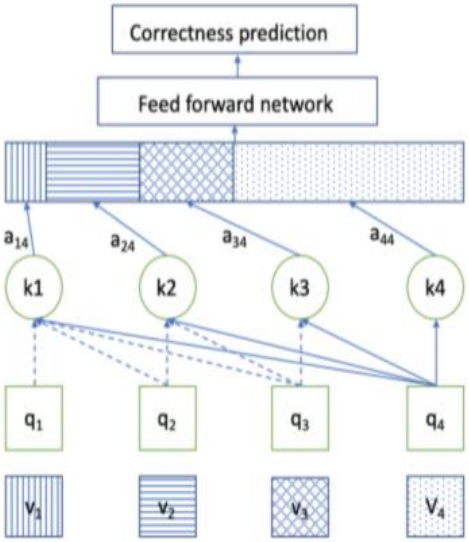

Figure 12: SAKT Framework. [34]

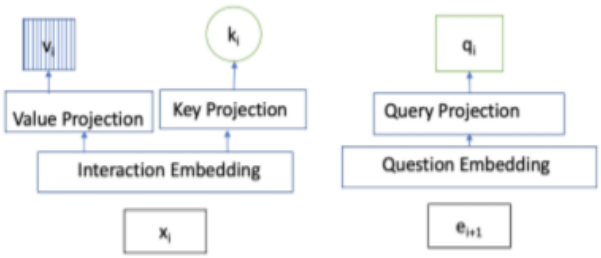

Figure 13: SAKT's Embedding layers. [34]

4.2.3 Graph KT models

Since the scope of this thesis is focused on dynamic machine learning models, a reference to some of the available Graph-based KT models will be valuable in determining the range of models that tried to predict the knowledge mastery of students. The Graph Knowledge Tracing (GKT) proposed in 2019 and by leveraging the potential of Graph Neural Networks applied them to the KT domain. The quite recent model tried to provide more interpretable predictions compared to existing KT models utilizing a different way to update the knowledge state of a student as they interact with a coursework. At first, the proposed graph knowledge structure is composed of nodes entail the exercises' concepts and edges that are defined from the dependencies among concepts. Following, the update of the student's knowledge state triggered when a student answers a question derived from a concept, updating also the neighbor nodes-concepts. [35] Finally, the authors presented the superiority of the GKT on two well-known educational datasets over the DKT and DKVMN among other graph-based and learning-based models.

In 2020, another framework released that borrowed ideas from a hierarchical exercise graph and combined them with sequential deep learning models. The Hierarchical Graph Knowledge Tracing (HGKT) model make use of a LSTM model and is an enhanced version of the model Exercise Enhanced Knowledge Tracing (EKT) which was the first model that utilized features from exercises in a Bidirectional LSTM model. HGKT assumes that if a student answers correctly an exercise also has higher probability to answer correct another exercise that shares a common schema. Common schema

is considered by the authors the common characteristics of a group of exercises. [36] Additionally, in the same paper they proposed different variants of the initial model. HGKT-B employ word embedding derived from the BERT pretraining method to encode text features and HGKT-S focuses on a one-hot representation of common schema. HGKT displays results greater than many KT models, such as BKT, DKT, DKVMN and GKT.

4.3 Synopsis of the related work

As can be observed from the above, there are numerous efforts in predicting the students' performance. The proposed categories of the Knowledge Tracing domain found in the literature have been extended through years leading to a long list of KT models. These models exhibit a sequential incremental performance on the corresponding papers constrained by the dataset's origin and format, as well as the technique of splitting the data to validate the performance of a model and the different libraries that are used to train the model. Thus, the performance of a model varies from paper to paper even though the evaluation metrics are common in every model. The focus in the literature was on the deep learning-based methods, on models that have extensively used in the last years as the main model of a paper or as part of baselines.

5 Methodology

This chapter is concentrated on the description of the roadmap to the final thesis model architecture and the full report of the general experiment settings such as the selection of the datasets and the evaluation metrics.

5.1 Datasets and Data Preprocessing

Due to the huge number of MOOCs enrollments, an even bigger number of data is generated through the students' interactions. The data contain a set of features that reveals the students' progress through time as they answer a series of questions. The logs data vary on each online platform but some attributes, such as student's id, skill's id, exercise's tag as well as the student's responses found common in every utilized dataset in the studied literature. Moreover, some KT models perform different preprocessing steps to the dataset to conform with the model's inputs or even to boost their performance by removing for example students with a single interaction [3].

5.1.1 Datasets

Dataset Selection and Format

The data format that has been used in KT task can be categorized into 'Columns format' and 'Three lines format'. The first differentiation consists of data in tabular form where each column is a separate feature produced by logs, while the second consists of data in the form of three lines. In the latter format, the first line is the summary of exercises a student attempted, the second line is the sequence of exercise tags, that is the skill ids and the third line is the sequence of responses to the corresponding exercises.

The datasets' selection to participate in the experiments was based on the availability of the skill's name of each problem that was answered by a student in the format of three lines. This restriction was set since all the different variations of the model of this thesis converted skill names to word embeddings of specific dimensions. Additionally, this

dataset format was also used by the most of KT models that this thesis model compares to.

Dataset Description

The presence of skill name mapping to skill id accompanied by the desired dataset format results in the final selection of four real-world datasets: 'ASSISTments2009_updated', 'ASSISTments2009_corrected', 'ASSITments2012_2013' and 'FSAI-F1toF3'. As it is observed, the first three datasets refer to the same online tutoring platform 'ASSISTments', but the second dataset is a corrected version of the first one and the third is newer version of the first. The summary of these datasets is depicted in Table 2 and each one is explained separately below. The correctness ratio was calculated for the test's set dividing the sum of students that correctly answered a question by the total number of answers. The correctness ratio can be regarded as the baseline AUC score for our model.

ASSISTments2009_updated: This dataset was generated from the 'ASSISTments' online learning platform and is one of the datasets that is extensively used in the KT task from several models. This dataset is also known as 'ASSISTment09' and contains data on mathematical problems from the school year 2009-2010. According to [37], on this dataset are detected data quality issues concerning duplicate rows.

ASSISTments2009_corrected: This dataset contains the corrections of the dataset 'ASSISTments2009_updated' that were reported for the first time in [38] and derived from duplicated records of a single student engagement. Practically, as mentioned in [2], a single question-and-answer interaction is associated with multiple skills allowing the DKT model to have access to the ground truth and increasing its performance. Moreover, these duplicate records formatted a new row to the dataset with a new skill tag that are handled differently from [2] in our preprocessing steps.

ASSISTments2012_2013: The volume of this dataset is bigger than 'ASSIST2009_updated' and it contains students' data from the school year 2012-2013 that were retrieved from ASSISTments 's skill builder problem sets. A student is regarded that has successfully mastered a skill if they answered correctly in a sequence of 3 questions. After a particular skill's mastery, the platform proceeds to the next skills questions.

FSAI-F1toF3: This dataset was made available by the 'Find Solution Ai Limited' and involves students' responses to mathematical problems from 7th grade to 9th grade in

Hong Kong gathered from an application named ‘4LittleTrees’. Even though the number of skills is 99, a KT model that is compared to the thesis model makes use of the 2.266 questions as skills to train their model as the questions' tag was provided in the dataset. We conform to this strategy to train our model in this dataset.

Table 2: Summary of datasets.

Dataset	Number of distinct			Correctness ratio
	Skills	Students	Interactions	
ASSISTments2009_updated	110	4.151	325.637	65.84%
ASSISTments2009_corrected	101	4.151	274.590	66.31%
ASSISTments2012_2013	196	28.834	2.036.080	69.65%
FSAI-F1toF3	99	310	51.283	47.02%

Dataset splitting

The datasets ‘ASSISTments2009_updated’ and ‘FSAI-F1toF3’ were retrieved from the comparison baselines’ official GitHub repositories. The splitting methodology that has been followed by them is clarified as follows; Each dataset was divided into 70% for training and 30% for testing. Then, the training set was split into 5 pairs of 80% for training and 20% for validation. The same splitting technique was applied by us to the datasets ‘ASSISTments2009_corrected’ and ‘ASSISTments2012_2013’ using each time a random split.

5.1.2 Data Preprocessing

Data preprocessing is a necessary part of Machine and Deep Learning problems as the data quality affects the ability of a model to function and finally learn. Consequently, we need to walk through some preprocessing steps before feeding data into our model.

The dataset ‘ASSISTments2009_corrected’ contained skills of the form of ‘skill1_skill2’ and ‘skill1_skill2_skill3’ that could not be mapped to their corresponding names. Rather than removing completely the records with the specific skill id formats, we have decided to assign the first skill id, found before the underscore, to the student’s skill at the particular timestep. In other words, the skill ‘10_13’ was replaced with skill ‘10’ and so on. Moreover, a few misspellings were observed that were corrected and the

punctuations found in three skill names were converted to the corresponding words. For example, in the skill name ‘Parts of a Polnomial Terms Coefficient Monomial Exponent Variable’ we corrected the ‘Polnomial’ with ‘Polynomial’ and in the name ‘Order of Operations +,-,/,* () positive reals’ we replaced ‘+,-,/,* ()’ with ‘addition subtraction division multiplication parentheses’. The latter preprocessing action was preferred over the removal of punctuations since the datasets referred to mathematical methods and operations and without them, we would produce different embeddings losing the meaning of each skill. The above procedure has been followed to the ‘ASSISTments2012_2013’ dataset that contained additionally two missing values in the column with the skill ids and the entire rows were discarded. Spaces after some skill names were removed as well to ensure the correct generation of word embeddings. For the ‘ASSISTments2009_updated’ dataset, the skill with id ‘163’was replaced with the skill id ‘85’ since they referred to the same skill name. Moreover, in the same dataset, it was observed that some skill names were misspelled, such as the ‘Effect of Changing Dimensions of a Shape Prportionally’. These spellings mistakes were corrected to produce the proper word embeddings. Finally, the dataset ‘FSAI-F1toF3’ remained untouched as were not observed any mistakes to be corrected.

5.2 Metrics

The evaluation metric that has been used to assess our model is the Area Under the ROC Curve (*AUC*). AUC score is extensively employed in the knowledge tracing literature and is suitable for the datasets with imbalanced distribution as well as for binary predictions. [1], [39] A 50% AUC score denotes that the model performs poorly and its predictions can be regarded as random guesses. On the contrary, a high AUC score reveals high prediction performance. In Figure 14 is presented a sample area under the ROC curve and the segmentation of a classifier depending on its position in the graph. A classifier with AUC scores formatting a line that is above the diagonal can be regarded as good while its closeness to the upper left point makes it optimal and most probable ready for business use.

Using the AUC metric of Keras API it is important to note its exact calculation in the source code. AUC computation requires the calculation of 4 variables known as True Positives, True Negatives, False Positives and False Negatives. Their definition is out of purpose of this thesis and were omitted. The ROC curve is calculated with the values of

the recall by the false positive rate setting a threshold, in this case is left to the default number of 200, which controls the discretization of the predictions. [40]

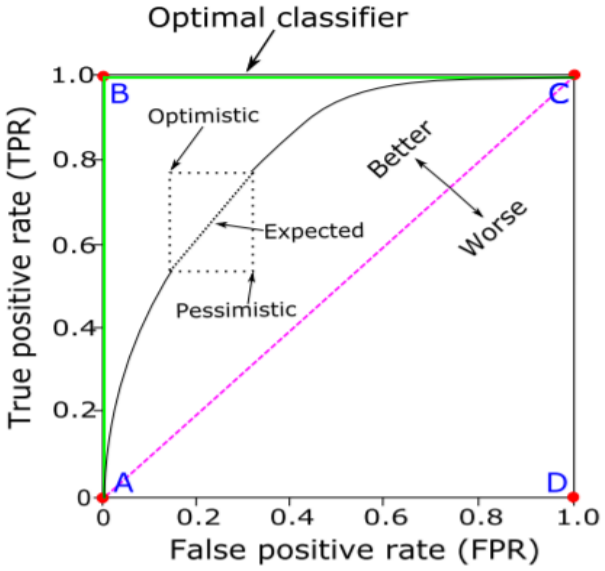

Figure 14: Sample AUC-based positioning and segmentation of a classifier. [41]

Additionally, the *binary cross-entropy* as the model's loss function and the *accuracy* have been also reported to establish our model's performance. The Mean Square Error (MSE) can also be found in the implementation of some KT models, such as in the DKT model, but the cross-entropy loss is regarded more suitable for binary classification tasks since the students' answers to coursework's questions were either correct or wrong.

5.3 Experiments

In the experimental phase of this dissertation, many architectures have been produced until we reach the final model. The criterion of the selection of the best model was its performance on the average validation AUC. More precisely, we train each model five times using each time one of the five previously mentioned training sets and measure its performance on the corresponding validation sets. Then, the average AUC was calculated and compared with those of the rest model versions to obtain the final architecture. This technique is known as cross-validation and is deployed to machine learning models with the purpose to estimate the ability of a model to previously unseen data. Since we involve four datasets in the experiments, the final model should be performing also better to at least two of them to achieve a good generalization performance. These datasets were the 'ASSISTments2009_updated' and 'ASSISTments2009_corrected'. The first

dataset was selected because it is present in almost all studied papers while the second is a new release that has not been tested thoroughly in the literature.

Before starting the experiments, we needed to reconstruct the data in such a way to feed them in the first layer of the model. Having the datasets in a horizontal format, we convert them into a vertical format where the first column is the student id, the second is the student's skill and the third is the responses to each skill's questions. Each train and validation matrix has been reconstructed into a Toeplitz matrix of length *L*, stacking horizontally the inverted matrix with the responses of each student from the previous time step. This created a history window of each student's past interactions by hiding in each timestep their response and filling with zeros their future ones. The parameter L was tested against three values and are explained below. A sample of the skill matrix is demonstrated in Table 3 where a 5 sized skills' history window was constructed from the initial skill ids. The rows in the table represent each student's skill id in a particular exercise. As can be observed, there are consecutive exercises derived from the same skill id such as the skill with id 5. We highlighted with yellow, blue and green the different timestep of the 5-sized history window. The L parameter was chosen, for simplicity's sake, at 3.

Table 3: Sample of implemented history window.

Initial skills format	Reconstructed skills with 5 window size		
1	1	0	0
2	2	2	0
3	3	3	3
4	4	4	4
5	5	5	5
5	0	5	5
5	0	0	5

5.3.1 Roadmap to final model architecture

The Functional API of Tensorflow's Keras framework had been chosen, over the Sequential API, in the experiments to ease the development of each model as it provides flexibility and more control in the selection of inputs and outputs at each layer. The ver-

sion of this library is shown in a table in conjunction with all the libraries used in this dissertation at the end of this chapter.

General experiment parameters

As a starting point, we need to define some parameters that would help us produce our first results. The *Adam* optimizer with a learning rate of 0.001 and a *batch size* of 32 was chosen for each model architecture. Moreover, for each model version, the *number of epochs* was set to 20 trying to build a framework with relatively fast training that could produce good results with fewer epochs than the baselines.

Roadmap

The first two layers of the initial model architecture consist of inputs of both skills and responses followed by an Embedding layer for each Input layer. The initial embeddings were generated from a random uniform distribution of 100 dimensions while has also been tested the 300 dimensions, the FastText embeddings and the Word2Vec embeddings, pre-trained on the English Wikipedia corpus, to acquire the initial weights of each skills' name. The random embeddings were produced from an interval with a lower limit the -1/(embeddings size)*L and upper limit 1/(embeddings size) *L. The limits were multiplied with the window size L since the model produced better results than without it. Additionally, the pre-trained Word2Vec embeddings were extracted from [42] selecting the 'enwiki_20180420 (window=5, iteration=10, negative=15)' file containing both the 100 and 300 dimensions. The variables inside the parentheses define the parameters by which the neural network was trained to produce the embeddings while the numbers after the underscore refers to the date of the training. The procedure that has been followed to produce the sentence embeddings was to map each word in the skill name to its corresponding Word2Vec and FastText embeddings and calculate the average embeddings. The skill name embeddings were also produced by the summary of each skill's word embeddings and were rejected from the experiments since it degraded the initial model's performance and it was considered that skill names with a bigger number of words would be favored over those with fewer words. Note that during the model training the weights of the Embedding layer were updated. Next, we fed the skill name embeddings into a Convolutional layer consisted of 100 filters, a kernel size of 3, a stride of 1 and a ReLU activation. The filters parameter was set to 300 when

we trained our model with word embeddings of 300 dimensions. With a Concatenate layer, we joined the output of the Convolutional layer with the questions' weights to produce a single tensor that would be fed to the LSTM layer. The LSTM layer was composed of 64 units while only the last time step hidden state was returned during training, meaning that the "return_sequences" parameter was set to "False". Finally, with the addition of a Dropout layer after the LSTM followed by two Dense layers, we reach the end of the initial model architecture. The second Dense layer with 1 unit and sigmoid activation function was used to make the binary predictions while the intuition behind the first Dense layer with 50 units and a ReLU activation was to facilitate the final predictions with an intermediate activation function which indeed increased the performance of the model. The Dropout layer with a small dropout rate of 0.2 was placed after the LSTM layer since it was observed that the model was suffer from overfitting in two folds. Overfitting occurs when the model fits the dataset very well resulting in high performance, but it has poor performance on a new dataset.

The validation set determined the previous combination of hyperparameters that were selected against different configurations. More precisely, the *L* parameter that defined the window size was set also to 10 and 20 instead of the final 50 as it was observed that decreased the performance of the initial model. Moreover, the 64 units in the LSTM were capable to provide good results against a smaller number such as 32. When we increased the kernel size and the stride in the Convolutional layer, accompanied by the previously mentioned hyperparameters of the other layers, resulted also in a model degradation. As mentioned above, the original size of the random embeddings was set to 100. All the above experiments in the parameters and hyperparameters of the first framework were tested under the 'ReduceLROnPlateau' callback that reduces after the first 10 epochs the learning rate by 0.1 when the validation AUC score was not improved.

Figure 15 provides the flow of the experiments, assigning to a different experiment a different model version. We did not version in the different values of the hyperparameters in the experiments and we focused mainly on model versioning that involved the implementation of different layers and callbacks. The comparison among versions is made as the arrows suggest, for example, the Model_v1.1 was compared with Model_v1.0 and the Model_v1.2 with Model_v1.1.

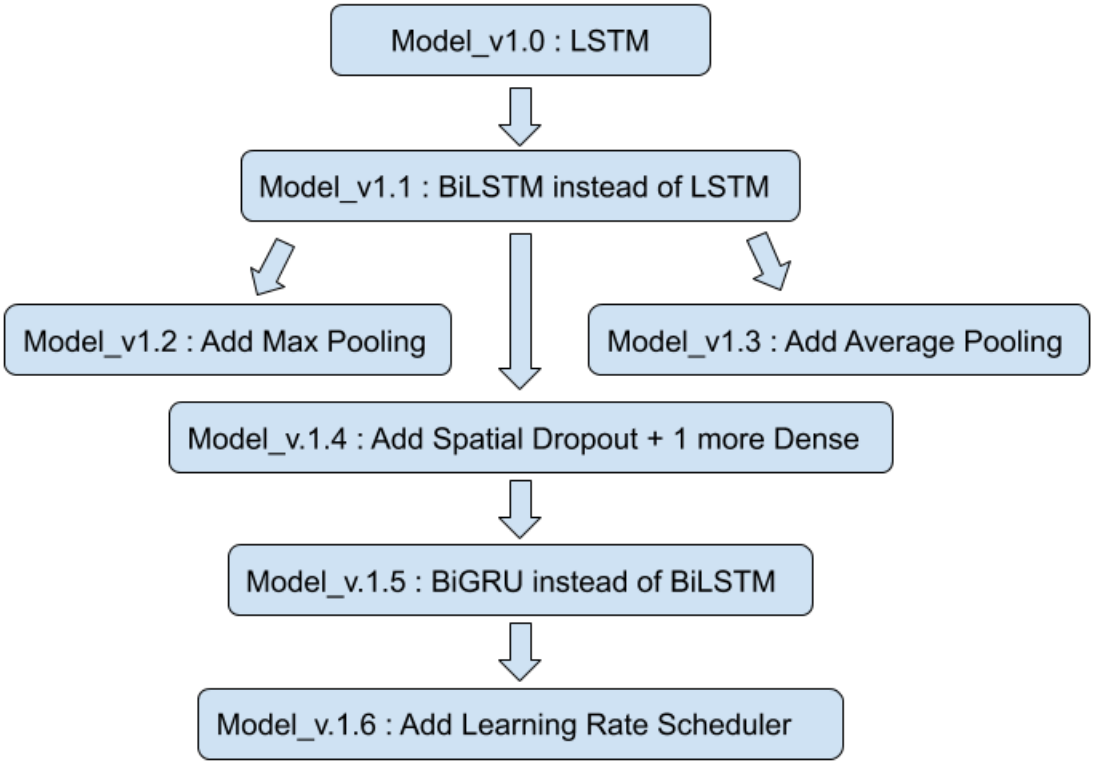

Figure 15: Roadmap to final model version.

In the version comparison, the percentage increase or decrease of the AUC score has been truncated into the third decimal point. The overall performance of each model version is depicted in Table 4 at the end of this subsection for convenience in the comparison of all metrics among versions. In the mentioned table, the AUC score and the accuracy are also presented truncated into the third decimal point while the loss in the fourth decimal. In each fold, every model's metric was truncated to the previous digit making the model enhancement process intentionally slightly more difficult.

When we changed the LSTM layer in Model_v1.0 with a Bidirectional LSTM layer we gain an increase of 3.078%.in 'ASSISTments2009_updated' and 4.471% in 'ASSISTments2009_corrected'. The bidirectional functionality has also been used in the KT literature by the EKTA model and it is believed that the resulting model, denoted as Model_v1.1, managed to better capture the student responses at each timestep since it traverses the sequences in two directions.

The next models, versioned as v1.2 and v1.3, utilized the pooling mechanism that is placed after the Convolutional layer. The addition of Max pooling layer formatted Model_v1.2 and the addition of Average pooling layer the Model_v1.3. Max pooling is basically a Nonlinear Sub-sampling method that places a building block after each Convolu-

tional layer extracting from each previous layer's block the maximum number. In conjunction with Max-pooling, the Average-pooling calculates the average of the previous layer's block. [23] In both added layers the pool size was left to its default value having a pool size of 2. A visual representation of a Sub-sampling layer can be seen in Figure 16 while in the Figure 17 is presented an example of a Max-pooling layer. The average validation AUC decreased in Model_v1.2 by 0.319% in 'ASSISTments2009_updated' and by 0.2255% in 'ASSISTments2009_corrected' compared to Model_v1.1. Similar behavior was noticed in Model_v1.3, in which the AUC was decreased from Model_v1.1 by 0.302% in 'ASSISTments2009_updated' and by 0.2633% in 'ASSISTments2009_corrected'.

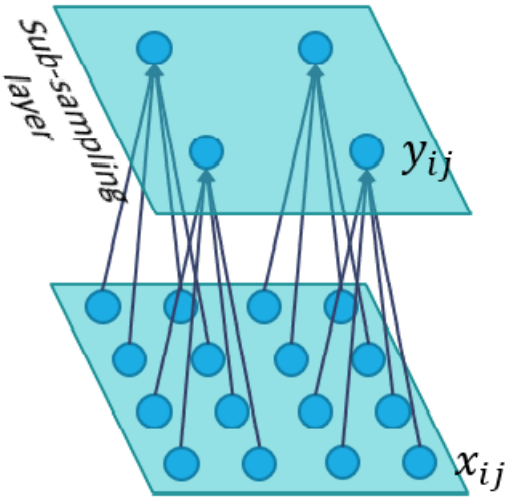

Figure 16: Sub-sampling layer. [23]

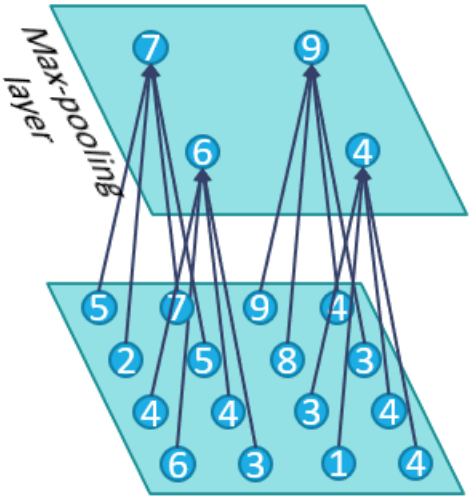

Figure 17: Max-pooling layer. [23]

Starting again from Model_v1.1, we added a SpatialDropout1D layer with a rate of 20% after each Embedding layer and another Dense layer with 25 units between the two Dense layers. As each layer addition individually improved slightly the performance, it has decided to be implemented simultaneously. We named the resulting model version as Model_v1.4 and the percentage increase in average validation AUC is 0.008% in 'ASSISTments2009_updated' and 0.014% in 'ASSISTments2009_corrected'. The SpatialDropout1D layer is similar to the Dropout layer but instead of randomly dropping elements as in Dropout, it drops the whole 1D feature maps. The intuition behind the addition of SpatialDropout1D after each embedding layer was the overfitting that was observed in the first epochs of each validation set and it was regarded that the correlation among skill name embeddings, that might not actually exist, confused the model.

The intermediate Dense layer was added for the same reason that the first Dense layer was added.

The next model version refers to the change of recurrent type; instead of using LSTM we alternated it to GRU. The Bidirectional layer remained also in this version since the performance of the model when this layer was added increased significantly the performance in both datasets. The resulting model, named as Model_v1.5, exhibited a percentage increase of 0.048% in 'ASSISTments2009_updated' and 0.021% in 'ASSISTments2009_corrected'.

Model_v.1.6, involved a technique to improve the performance of Model_v1.5. In Model_v1.6 we replaced the 'ReduceLROnPlateau' callback with a custom Learning Rate Scheduler callback function. Learning Rate Schedule is a method to alternate the learning rate of a neural network to fast training and often increase the performance of a model. We constructed a function that takes as inputs the epoch index and the initial learning rate and returns as output a new learning rate that has been decreasing exponentially after the first ten epochs. The mathematical formula that has been used is the following, where the lr is the learning rate and n is the number of epochs:

$$lr = lr \times e^{(0.1\times(10-n))} \quad (5)$$

The Model_v1.6 got a raise of 0.024% in 'ASSISTments2009_updated' and 0.025% in 'ASSISTments2009_corrected'compared to Model_v.1.5.

Finally, we alternated Model_v.1.6 with three separate modifications. The first one was the replacement of the Convolutional layer (Conv1D) with the LocallyConnected1D layer. The difference between this layer and the Conv1D layer is that the weights are not shared to inputs between the filters. The performance of the model was decreased by 1.807% in the 'ASSISTments2009_updated' and thus we excluded it to the next experiment. The second one was the addition of a Batch Normalization layer after the Dropout layer. Batch Normalization (BN) is a technique that is used to standardize the inputs of the next layer of the network. When training a model with batch normalization, it will scale the data to zero mean and standard deviation of one using the parameters 'beta_initializer' and 'gamma_initializer' that kept in our case to their default values. This layer also used to boost training speed and make it more stable. [43] The aforementioned addition led to a 0.026% decrease in the 'ASSISTments2009_updated' and hence we rejected this modification. Lastly, we replaced in the model the Concatenate layer with an attention layer. The implementation of this layer required two tensors and in

order to integrate it in our model, we replaced it with the concatenation layer. The first attention layer was the Dot-Product attention, also referred to as 'Attention' in the Tensorflow library, and it decreased the performance of the final model by 0.733% in 'ASSISTments2009_updated'. The percentage decrease was considered by us quite big to test it also in the 'ASSISTments2009_corrected'. The other attention layer that has been tested against the previous was the Additive Attention layer. We followed the example provided by the Tensorflow library that use this attention layer after two Convolutional layers. Hence, we replaced also in this case the Concatenate layer with the AdditiveAttention layer and the average validation AUC was decreased by 0.652% in 'ASSISTments2009_updated'. Since the average validation AUC was decreased in the last two modifications of the Model_v1.6, we discarded them as well. All the mentioned tries to improve the final model, due to their impact in the final versioned model, did not participate in the model versioning in Table 4.

Table 4: Model performance among versions.

Dataset	Model version	Average Validation		
		AUC	Loss	Accuracy
ASSISTments2009_updated	v1.0	79.530	0.4889	75.950
	v1.1	81.978	0.4617	77.174
	v1.2	81.716	0.4649	76.988
	v1.3	81.730	0.4638	77.057
	v1.4	81.985	0.4614	77.198
	v1.5	82.082	0.4611	77.222
	v1.6	82.102	0.4625	77.333
ASSISTments2009_corrected	v1.0	70.870	0.5690	71.556
	v1.1	74.039	0.5499	72.676
	v1.2	73.872	0.5507	72.619
	v1.3	73.844	0.5509	72.578
	v1.4	74.050	0.5493	72.686
	v1.5	74.066	0.5492	72.716
	v1.6	74.085	0.5493	72.704

A clearer picture of how each model version performed across the folds for both 'ASSISTments2009_corrected' and 'ASSISTments2009_updated' can be seen in figures 18 and 19, respectively. From the first figure, we observe that each model version exhibits the same pattern in the validation AUC score. The lowest AUC achieved by all versions in the second fold while in the third fold each one gained the highest.

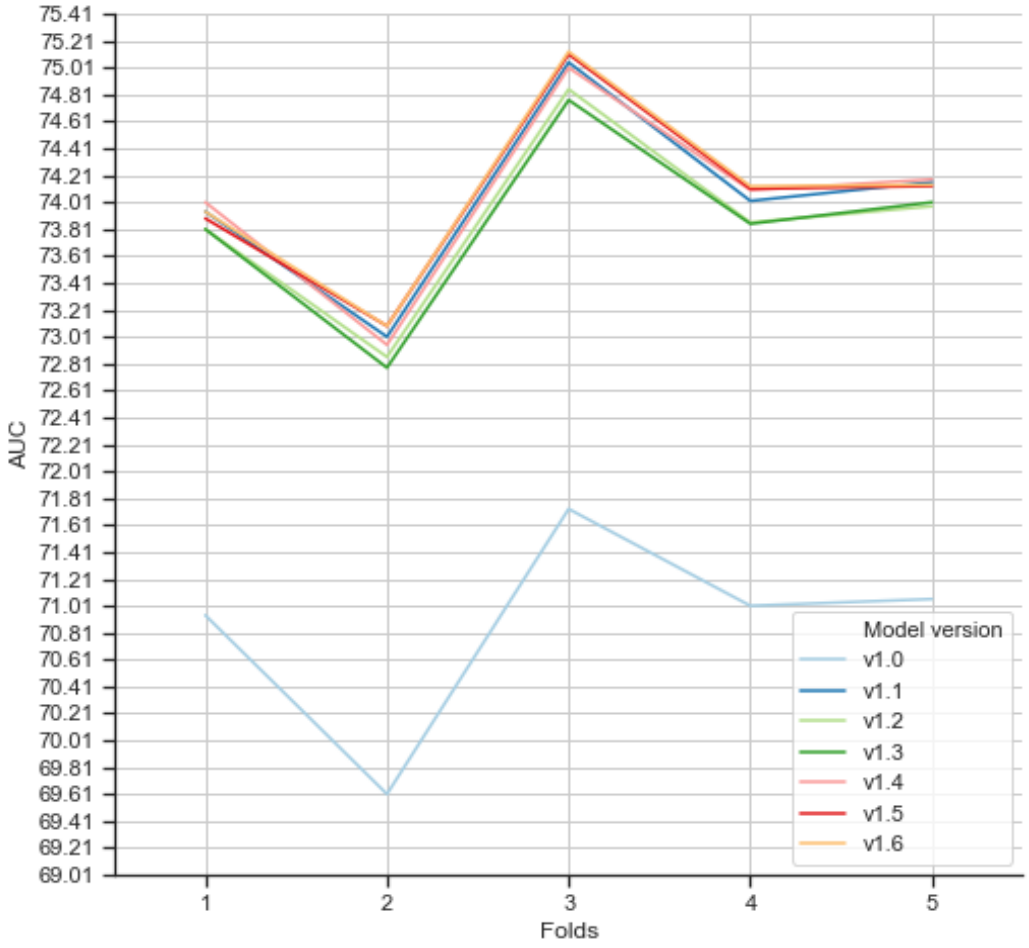

Figure 18: Validation AUC per fold in the 'ASSISTments2009_corrected '.

In the second figure (Figure 19), the Model_v1.0 presents the lowest validation AUC score in the third fold on the contrary with the first versioned model in Figure 18 in the same fold. The rest versions of the same figure present the same fluctuations with a different degree in the first three folds. More specifically, the highest validation AUC score was achieved in the second epoch in the models with versions 1.1, 1.2, 1.3, 1.4, 1.5 and 1.6 and the lowest in the models with versions 1.0, 1.1, 1.3, 1.4, 1.5 and 1.6 in the third fold of the 'ASSISTments2009_updated'.

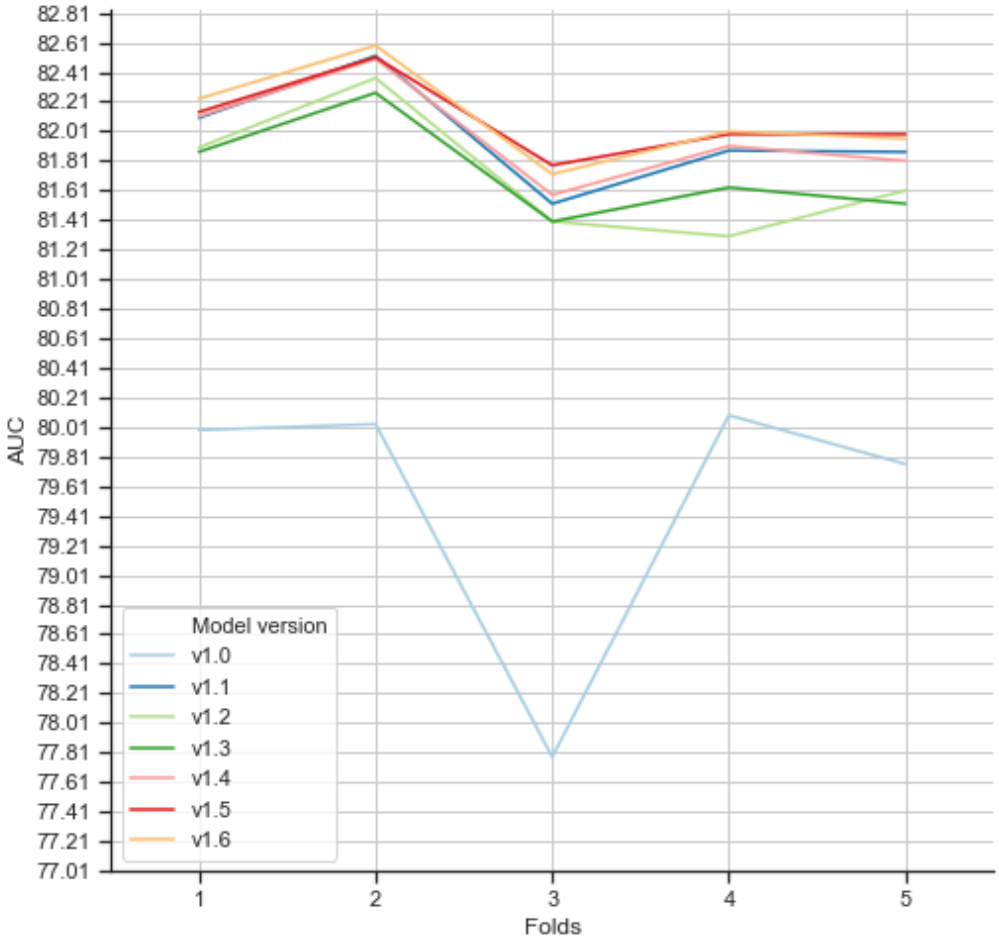

Figure 19: Validation AUC per fold in the 'ASSISTments2009_updated '.

5.3.2 Final model

The final model architecture derived from the previous experiments is depicted in Figure 20. The parameters and the hyperparameters of this framework remained the same as mentioned above while in the algorithm we constructed several functions needed to properly operate, such as loading and reformatting the data, training and testing the underlying model. The thesis model was uploaded in a GitHub repository for review and testing and can be found in [44].

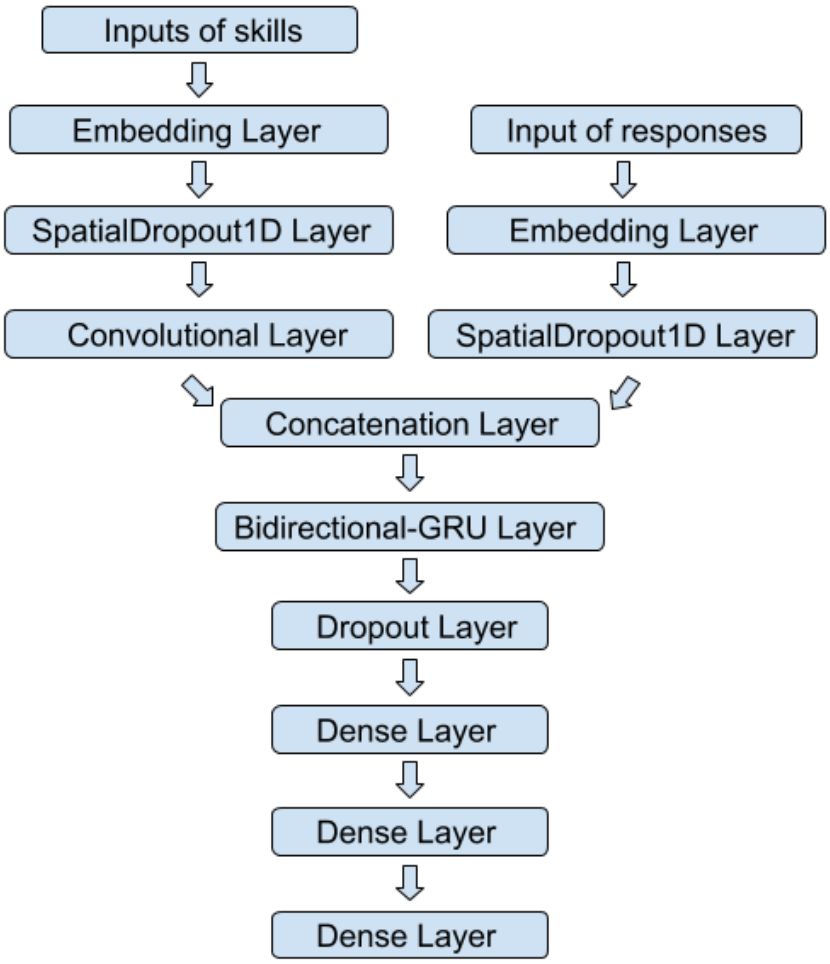

Figure 20: Final model architecture.

5.4 Experiments Environment and Libraries

Every baseline that participated in the comparison with our model utilized a different version of libraries to run their experiments. In some papers, are clarified the characteristics of the computer in which they run their KT model. For example, in the 'Implementation' section of [1], the authors mention that used the Tensorflow library and run their experiments on a computer with NVIDIA RTX 2080 GPU. Thus, our experiments' environment along with the selected libraries are presented below providing full transparency of the results.

5.4.1 Experiments Environment

Two environments have been selected to conduct our experiments. The first one was the local machine in which we ran the algorithms of the competitors and the second was the Google Colaboratory in which we ran our experiments. Google Colaboratory is a free cloud service in which you can basically write and execute code with many more other

capabilities. This discrimination was considered necessary as we performed a series of experiments in different combinations of hyperparameters and layers until we reached the final thesis algorithm and is undoubtedly a time-consuming procedure. We enabled GPU as a hardware accelerator of Google Colaboratory to minimize the time of each experiment and leveraged its connection with Google Drive to save local space since we utilized four datasets, some of which were quite big in size. Nonetheless, for the sake of comparison with the baselines, we ran the final model version also in our local machine and no significant discrepancies were observed in both cross validation and testing. The specifications of the software can be seen in Table 5.

Table 5: Specifications of our local machine.

Operating system name	Microsoft Windows 10 Home
System manufacturer	HP
Version	10.0.18363 Build 18363
Processor	AMD Ryzen 5 2500U
Graphics	AMD Radeon™ Vega 8 Graphics
RAM	4 GB

5.4.2 Experiments Libraries

The major libraries accompanied by their versions and their purpose in our experiments are depicted in Figure 21. These libraries are open source and are extensively used by the Python community for data analysis and machine learning tasks.

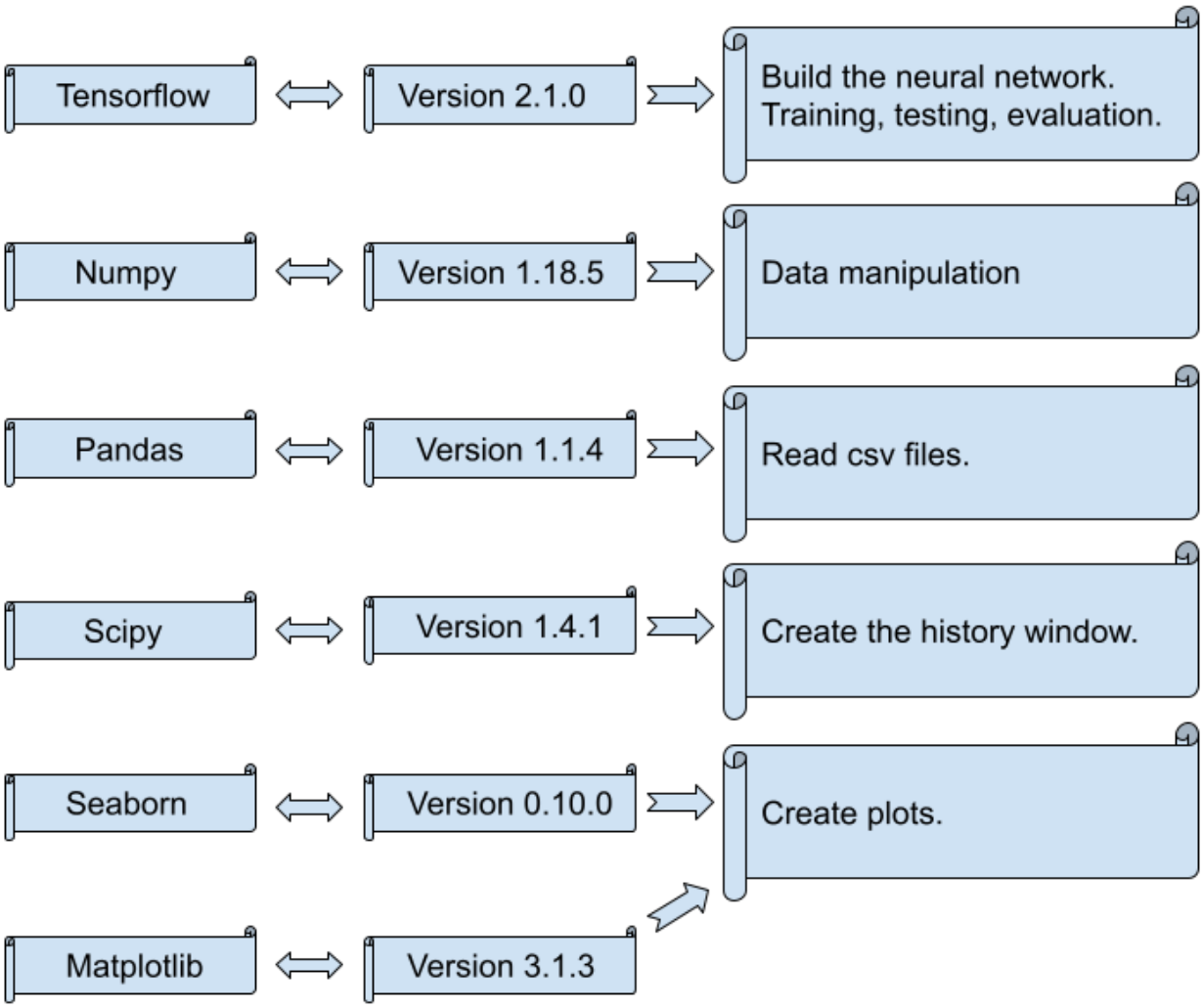

Figure 21: Libraries used in the experiments.

6 Results and Comparison with baselines

In this chapter are demonstrated the results of both our model and the competitors' Knowledge Tracing model followed by a comparison among them.

6.1 Results

The final model architecture has been tested in word embeddings derived from a random uniform distribution, Word2Vec and FastText embeddings. The dimensions of the first two were 100 and 300 while the third was the 300 dimensions. We trained the final model in 70% of the dataset and test it in the rest 30%. This method is known as Hold-out and is commonly used to estimate the performance on unseen data, usually smaller in size, with a single run. In the Table 6 are shown the thesis model's test results in AUC score, Accuracy and Loss of all datasets participated in this thesis.

As we observe in the table, the performance of the model increased with the use of Word2Vec embeddings of 100 dimensions as initial weights in 'ASSIST-ments2009_updated', 'ASSISTments2012_2013' and 'FSAI-F1toF3' by 0.060%, 0.506% and 0.994% respectively, compared to the random embeddings of 100 dimensions. On the contrary, in the dataset 'ASSISTments2009_corrected', the model's test AUC decreased by 0.054% in Word2Vec embeddings of 100 dimensions. The model in 300 dimensions displays a difficulty to predict the answers of each student in all datasets. This phenomenon worsens in almost all datasets in the replacement of Word2vec embeddings of 300 dimensions instead of the random embeddings of the same dimensions. More precisely, the model displays a percentage decrease by 0.004%, 0.013% and 0.623% with the use of Word2Vec embeddings of 300 dimensions in the 'ASSIST-ments2009_updated', 'ASSIST-ments2009_corrected' and 'FSAI-F1toF3' respectively. In the latter case, the model gained 0.138% in the 'ASSIST-ments2012_2013'. The effect of 300-dimensioned FastText embeddings in the model was also mixed. In the datasets 'ASSISTments2009_updated', 'ASSIST-ments2012_2013' and 'FSAI-F1toF3'

the percentage decrease was 0.120%, 0.088% and 0.851% respectively while in the dataset 'ASSIST-ments2009_corrected' was increased by 0.188% compared to also Word2Vec embeddings of the same dimensions.

Table 6: Evaluation of the final model in different embeddings and dimensions.

Dataset	Emb. Size	Emb. Type	Test AUC	Test Loss	Test Accuracy
ASSISTments2009_updated	100	Random	82.481	0.4581	77.370
	100	Word2Vec	82.531	0.4567	77.393
	300	Random	82.434	0.4558	77.458
	300	Word2Vec	82.430	0.4573	77.274
	300	FastText	82.331	0.4583	77.303
ASSISTments2009_corrected	100	Random	75.087	0.5427	73.271
	100	Word2Vec	75.046	0.5419	73.242
	300	Random	75.062	0.5421	73.197
	300	Word2Vec	74.931	0.5423	73.135
	300	FastText	75.072	0.5416	73.305
ASSISTments2012_2013	100	Random	67.331	0.5725	71.103
	100	Word2Vec	67.672	0.5706	71.223
	300	Random	65.584	0.5801	70.853
	300	Word2Vec	65.675	0.5798	70.844
	300	FastText	65.617	0.5798	70.877
FSAI-F1toF3	100	Random	65.953	1.2330	63.465
	100	Word2Vec	66.609	1.2087	63.827
	300	Random	65.962	1.3496	63.503
	300	Word2Vec	65.551	1.4402	62.594
	300	FastText	64.993	1.4462	62.609

6.2 Comparison with baselines

The KT models by which we compared the thesis model, as mentioned above, were the DKT, DKVMN and the DEEP-IRT model. Their GitHub repositories can be found individually in [45], [46] and [47]. In the corresponding paper of the two latter models, they are compared with the DKT model implemented by them following the details provided in the DKT's paper. Nonetheless, the original GitHub repository of DKT is written in the Lua programming language and the authors of the paper did not change the RNN to LSTM although the LSTM's results are provided in their paper. Hence, we decided to provide the results of the DKT that are reported in the DEEP-IRT model's paper, as it is an extension of DKVMN with similar results in their paper and the results that were produced by us from a GitHub repository that contains the DKT model in Python programming language.

In Table 7 one can see the overall performance of all models in which the 'Thesis Model' denotes our model, 'DKT_1' denotes the DKT model reported in the DEEP-IRT's paper and 'DKT_2' refers to our run in DKT. Moreover, the dash symbol in the columns of metrics denotes that the respective model did not involve this dataset in the paper and thus the results are missing. The values in the 'Emb.' column refer to the type of embeddings and they are accompanied by their dimensions. The displayed results are derived from our model's best AUC score in each dataset and are derived from Table 6. Finally, as the results of the competitor's model produced rounded in the second decimal point, we will follow the same for the thesis model.

From the table below we observe that the model constructed for this thesis outperforms in test AUC score the DKT_1 by 0.97%, the DKVMN by 1.33% and the DEEP_IRT by 1.15% in the 'ASSISTments2009_updated'. Additionally, in the 'ASSISTments2009_corrected' our model demonstrates an increase in test AUC of 1.27% compared to DKVMN and 1.68% compared to DEEP-IRT. In both mentioned datasets, the test accuracy of the thesis model is the highest and the loss is the smallest among the rest KT models. In the datasets 'ASSISTments2012_2013' and 'FSAI-F1toF3', the test AUC of our model is lower than the baseline models. More precisely, in the 'ASSISTments2012-2013', the difference in test AUC is 1.57% and 2.06% lower compared to the DKVMN and DEEP-IRT, respectively. In the 'FSAI-F1toF3', our model falls behind the DKT_1 by 2.81%, the DKVMN by 2.33% and the DEEP-IRT by 0.6% in test AUC.

Notable is the fact that the model DKT_2 presents a strange behavior in test AUC when we run the model multiple times. The developer of this model shuffles the dataset before it splits it randomly to training and testing sets. Therefore, every time we were running the model, we were producing results with large deviations. We decided not to change the source code in this thesis, as the authors of the model DKVMN and DEEP-IRT provided in their papers the results of the DKT model and might be derived from the same source code that was chronologically pre-existing. The DKT_2 outperforms our model by 2.75% in the 'ASSISTments2009_updated' and our model exceeds DKT_2 by 2.8% in 'ASSISTments2009_corrected'. Additionally, the DKT_2 has the greatest test AUC score among the rest models in the dataset ‘ASSISTments2012_2013’ leaving behind our model by 4.53%. The same happens for the ‘FSAI-F1toF3’ in which the DKT_2 exceeds in AUC our model by 2.81%.

Table 7: Comparison of thesis's model with baselines.

Dataset	Model	Emb.	Test		
			AUC (%)	Loss	Accuracy (%)
ASSISTments2009_updated	Thesis Model	Word2Vec 100 dim.	82.53	0.457	77.39
	DKT_1	-	81.56	0.526	77.17
	DKT_2	-	85.28	0.545	79.78
	DKVMN	-	81.20	0.471	76.71
	DEEP-IRT	-	81.38	0.532	76.91
ASSISTments2009_corrected	Thesis Model	Random 100 dim.	75.09	0.543	73.27
	DKT_1	-	-	-	-
	DKT_2	-	72.29	0.074	73.79
	DKVMN	-	73.82	0.553	72.56
	DEEP-IRT	-	73.41	0.631	72.59
ASSISTments2012_2013	Thesis Model	Word2Vec 100 dim.	67.67	0.571	71.22
	DKT_1	-	-	-	-
	DKT_2	-	72.20	0.092	69.94
	DKVMN	-	69.24	0.562	71.72
	DEEP-IRT	-	69.73	0.064	72.02
FSAI-F1toF3	Thesis Model	Word2Vec 100 dim	66.61	1.209	63.83
	DKT_1	-	69.42	0.826	64.11
	DKT_2	-	68.74	0.246	63.960
	DKVMN	-	68.94	0.631	63.60
	DEEP-IRT	-	67.21	0.870	62.24

7 Conclusions

This thesis tried to solve the students' performance prediction problem by using various types of Recurrent Neural Networks. The subject can be considered as a Knowledge Tracing task and has been studied extensively for many years by various researchers. These scientists contributed to the education sector by providing valuable models with exponential performance improvement, competing with the previous models even in the decimal points. This latter observation pushed this work to focus on the percentage decreases or increases between the models' versions and the differences, mostly on AUC score, among our model and the participated KT models.

This work utilized datasets that exceeded the number of datasets found in published papers, such as the [33] and the [48] that involved only one dataset that did not meet in 'popular' KT papers. These datasets were sufficiently big in size with some of them to contain millions of students' interactions in online exercises. The big-sized datasets are necessary for Machine and Deep Learning tasks to accurately predict the objective target. Hence, the need to train fast those datasets led to run the experiments in another source except the local machine and that is the Google Colaboratory.

The preprocessing step was a necessary part before we started the experiments as we utilized word embeddings that should be produced correctly. It seems that the NLP domain is highly attached to the KT domain, as many recent models involve embeddings in their models, such as those found in papers [1], [2] and [4]. The performance improvement of those models over the prior KT models might suggest that the word embedding contributed to the KT task in conjunction with the new model architectures. Moreover, many KT models integrated attention mechanisms trying to improve the AUC score and capture the information of dependencies among skills or exercises that are far away in time when a student solving them. Some models with attention mechanisms are DKVMN and DEEP-IRT and those that are found in papers [4] and [34]. In the current thesis, we provided results derived from two attention layers, the Dot-product and the Additive attention, and the percentage decreases presented in Chapter 5 can only mean that they failed to improve the performance of our final model. The referred experiments question the suitability of attention mechanisms in conjunction with

the rest layers of our model and especially with the Convolutional layer. The Convolutional layer has not been tested in the KT domain by any model studied in the literature.

As for the training time of our model, it was considered redundant to be studied in this thesis because in real-life applications a model is trained in advance and the evaluation of unseen data happens with relatively fast speed.

During the process, we decided to include an implementation of a DKT model found available with the desired programming language that was also part of a Capstone Project in a Udacity course as the competitors provide some unknown implementation of the DKT model. As can be observed from Chapter 6, we did not pay significant attention to the results that were produced by this model as the purpose was the exploration of an implementation of the DKT model because the original code was not available from the authors even if this model was a breakthrough in the KT problem.

Deep Learning-based KT models have been criticized for different reasons mentioned in the introduction, mostly about their interpretation and overfitting in the KT datasets. Overfitting is a problem that is also observed by us and we tried to tackle it in this dissertation. Nevertheless, these models exceed in test AUC many models that use other approaches, such as the Traditional KT models. Deep Learning models still exhibit a moderate performance in the KT domain compared to other tasks like Machine Translation or Speech Recognition and that can be seen in the results presented in Table 7.

The model produced for this thesis was a result of many versions and hyperparameter selection and achieved greater results in two out of four datasets, in at least three out of four denoted models. We conclude that DKT_1 and DKT_2 can be regarded as a different model for the reasons mentioned above. The percentage difference in test AUC in 'ASSISTments2009_updated' and 'ASSISTments2009_corrected 'is noticeable among our model and the models DKVMN and DEEP-IRT. Nonetheless, our model failed to provide better results in ‘ASSISTments2012_2013’ and ‘FSAI-F1toF3’. The latter can be justified as the number of skills in the first two datasets was close to 100 but in the last two was almost twice and twenty-three times bigger. This could mean that our model is more suitable for KT datasets with a small number of skills. Either way, this thesis exposes a new and untested architecture that provides interesting results and could be a subject for further improvements.

8 Future work

The recently developed models that participated in the experiments of this thesis and the models that keep submitting to the conference of Educational Data Mining (EDM) validate the need to exceed the existing KT models in performance. By approaching the knowledge tracing problem with Deep Learning methods and especially with Recurrent Neural Networks started a new era that attracted much interest. The continuous enhancements in performance suggest that with integrations and modifications there is still room for improvements.

Worth mentioning is the use of an enhancement version of MANN as an attention mechanism of DKVMN that improved its performance and exceeded the DKT and the union of the IRT with DKVMN which resulted in a new model, named DEEP-IRT, that provided more explainability while retained the performance in the results presented in the corresponding paper. The latter gives us the idea that instead of training the thesis model at a skill level, someone could make use of the problem id and build an IRT model benefiting from the advantages of item response theory. Additionally, there are other attributes in the datasets that one can employ in our model, such as the teacher id and the predictions could be made complimentary by tracking the students derived from the cluster of teachers. The latter has been taken into consideration but for comparison purposes with the rest of the KT models that were trained in datasets with specific split and format, constrained by particular attributes, we rejected the initial idea.

The current model can also be further extended by using different embeddings such as the Bert embedding that found in [36] and other untested embeddings like the ELMo and BigBird that use bigger number of dimensions. Due to limitations in the use of GPU in Google Colaboratory, this was infeasible in our case. Even though the results in this thesis shown that the bigger number of dimensions degraded the performance of our model, different embeddings with higher dimensions would establish more our conclusions.

Another recommendation for future work would be the hyperparameter optimization of our model that could be achieved with the use of the ‘GridSearchCV’ module of the scikit-learn machine learning library. A fine-tuning of a Deep Learning model is a difficult procedure due to the big number of hyperparameters of each layer that can even get

larger due to the depth of each models' architecture. In this thesis, we also tried to implement a grid search into the hyperparameters' space but due to limitations in the maximum Virtual Machine's lifetime and idle timeouts of Google Colaboratory's notebook, in which we run our experiments with GPU accelerator, led to a dead end and we turned our attention in the manual hyperparameter optimization.

Finally, since all datasets contained student’s answers on mathematical exercises, we believe that would be interesting for the thesis model to be trained in datasets with different content that might involve theoretical and grammar exercises of other courses like the English Language.

Bibliography

[1] Yeung, C.-K. 2019. ‘Deep-IRT: Make Deep Learning Based Knowledge Tracing Explainable Using Item Response Theory’. arXiv:1904.11738 [cs, stat].

[2] Liangbei Xu, Mark A. Davenport. 2020. ‘Dynamic Knowledge Embedding and Tracing’. arXiv: 2005.09109v1 [cs.IR].

[3] Chris Piech, Jonathan Bassen, Jonathan Huang, Surya Ganguli, Mehran Sahami, Leonidas Guibas, and Jascha Sohl-Dickstein. 2015. ‘Deep Knowledge Tracing’. In Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, Montreal, Quebec, Canada (NeurIPS). Cambridge, MA, USA,505–513.

[4] Abdelrahman, G., Wang, Q., 2019. ‘Knowledge Tracing with Sequential Key-Value Memory Networks’. Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval- SIGIR’19 175–184.

[5] Jiani Zhang, Xingjian Shi, Irwin King, and Dit-Yan Yeung. 2017. ‘Dynamic key-value memory networks for knowledge tracing’. In Proceedings of the 26th International Conference on World Wide Web, pages 765–774. International World Wide Web Conferences Steering Committee.

[6] W. J. Zhang, G. Yang, Y. Lin, C. Ji and M. M. Gupta. 2018. ‘On Definition of Deep Learning’. World Automation Congress (WAC), Stevenson, WA, 2018, pp. 1-5, DOI: 10.23919/WAC.2018.8430387.

[7] Mohammad M. Khajah, Robert V. Lindsey, and Michael C. Mozer. 2016. ‘How deep is knowledge tracing’. In Proceedings of the 9th International Conference on Educational Data Mining, pages 94–101.

[8] Haohan Wang, Bhiksha Raj. ‘On the Origin of Deep Learning’. 2017. Language Technologies Institute School of Computer Science Carnegie Mellon University. arXiv:1702.07800 [cs.LG].

[9] Kasem Khalil, Omar Eldash, Ashok Kumar, Magdy Bayoumi. 2018. ‘An Efficient Approach for Neural Network Architecture’. DOI:10.1109/ICECS.2018.8617887.

[10] ELLIOT, A. J., AND DWECK, C. S. ‘Handbook of competence and motivation’.2013. Guilford Publications.

[11] Christopher James Piech. 2015. ‘Uncovering Patterns in Student Work: Machine Learning to Understand Human Learning’. Ph.D. Dissertation. Stanford University.

[12] S. Hochreiter and J. Schmidhuber. ‘Long short term memory’. 1997. Neural computation 9, 8:1735–1780.

[13] Original paper of LSTM: ‘http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.4320&rep=rep1&type=pdf”.

[14] S. Hochreiter and J. Schmidhuber, ‘Long Short-Term Memory’. Neural Computation 9, 8.

[15] ‘https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21’.

[16] Cho Kyunghyun, van Merrienboer Bart, Gulcehre Caglar, Bahdanau Dzmitry, Bougares Fethi, Schwenk Holger, Bengio Yoshua 2014.’Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation’. arXiv:1406.1078.

[17] Ravanelli Mirco, Brakel Philemon, Omologo Maurizio, Bengio Yoshua. 2018. ‘Light Gated Recurrent Units for Speech Recognition’. IEEE Transactions on Emerging Topics in Computational Intelligence, 2: 92–102. arXiv:1803.10225. doi:10.1109/TETCI.2017.2762739.

[18] Su Yuanhang, Kuo, Jay. 2014. ‘Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling’. arXiv:1412.3555.

[19] Schuster, Mike, and Kuldip K. Paliwal. 1997. ‘Bidirectional recurrent neural networks’. Signal Processing, IEEE Transactions on 45.11.

[20] Dernoncourt, Franck, Lee Ji Young, Szolovits Peter. 2017. ‘NeuroNER: an easy-to-use program for named-entity recognition based on neural networks’.arXiv:1705.05487.

[21] Sundermeyer, Martin, et al. 2014. ‘Translation modeling with bidirectional recurrent neural networks.’ Proceedings of the Conference on Empirical Methods on Natural Language Processing.

[22] Bi-LSTM: ‘https://medium.com/@raghavaggarwal0089/bi-lstm-bc3d68da8bd0’.

[23] K. Diamantaras, Advanced Machine Learning, Topic: ‘Deep Learning’. April 2020. Data Science. International Hellenic University. Thessaloniki.

[24] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, Kaori Togashi. 2018. ‘Convolutional neural networks: an overview and application in radiology’.

[25] C. Guo, F. Berkhahn. 2016. ‘Entity embeddings of categorical variables’. arXiv preprint arXiv:1604.06737.

[26] C. Berberidis, Natural Language Processing and Text Mining, Topic: ‘Word Embeddings’. March 2020. Data Science. International Hellenic University. Thessaloniki.

[27] Piotr Bojanowski, Edouard Grave, Armand Joulin, Tomas Mikolov. 2016. ‘Enriching Word Vectors with Subword Information’. arXiv:1607.04606v2 [cs.CL].

[28] Minh-Thang Luong, Hieu Pham, Christopher D. Manning. 2015. ‘Effective Approaches to Attention-based Neural Machine Translation’. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1412–1421.

[29] Albert T. Corbett and John R. Anderson. 2015. ‘Knowledge tracing: modeling the acquisition of procedural knowledge’. User Modeling and User-Adapted Interaction, 4 (4):253–278.

[30] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston. ‘Key-value memory networks for directly reading documents’. 2016. arXiv preprint arXiv:1606.03126, 2016.

[31] Z. A. Pardos and N. T. Heffernan. 2010.‘Modeling individualization in a bayesian networks implementation of knowledge tracing’. In User Modeling, Adaptation, and Personalization, pages 255–266. Springer.

[32] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. 2016. ‘Meta-learning with memory-augmented neural networks’. In Proceedings of the 33rd International Conference on Machine Learning, pages 1842–1850.

[33] Zhenya Huang, Yu Yin, Enhong Chen, Hui Xiong, Yu Su, Guoping Hu. 2019. ‘EKT: Exercise-aware Knowledge Tracing for Student Performance Prediction’. IEEE Transactions on Knowledge and Data Engineering.

[34] Shalini Pandey, George Karypis. ‘A Self-Attentive model for Knowledge Tracing’. 2019. arXiv:1907.06837v1 [cs.LG].

[35] Hiromi Nakagawa, Yusuke Iwasawa, and Yutaka Matsuo. ‘Graph-based Knowledge Tracing: Modeling Student Proficiency Using Graph Neural Network’. 2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI). IEEE, 156–163.

[36] HanshuangTong, Yun Zhou, Zhen Wang. 2020. ‘HGKT: Introducing Problem Schema with Hierarchical Exercise Graph for Knowledge Tracing’.arXiv:2006.16915v1 [cs.CY].

[37] ASSISTments09:‘https://sites.google.com/site/assistmentsdata/home/assistment-2009-2010-data/skill-builder-data-2009-2010’.

[38] K. H. Wilson, Y. Karklin, B. Han, and C. Ekanadham. 2016. ‘Back to the basics: Bayesian extensions of irt outperform neural networks for proficiency estimation’. arXiv preprint arXiv:1604.02336.

[39] Benjamin Clavié, Kobi Gal. 2020. ‘Deep Embeddings of Contextual Assessment Data for Improving Performance Prediction’. Proceedings of the 13th International Conference on Educational Data Mining.

[40] AUC score:’https://www.tensorflow.org/api_docs/python/tf/keras/metrics/AUC’

[41] Tharwat A. 2018.’Classification assessment methods’. Applied Computing and Informatics. doi:10.1016/j.aci.2018.08.003.

[42] Word2Vec emb.: ‘https://wikipedia2vec.github.io/wikipedia2vec/pretrained/’

[43] S. Ioffe, C. Szegedy. 2015. ‘Batch normalization: Accelerating deep network training by reducing internal covariate shift’. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML’15, page 448–456. JMLR.org.

[44] Thesis model in GitHub: ’https://github.com/GeorgiosChrysogonidis/Thesis-model’.

[45] DKT model: ‘https://github.com/lccasagrande/Deep-Knowledge-Tracing’.

[46] DKVMN model: ‘https://github.com/jennyzhang0215/DKVMN’.

[47] DEEP-IRT model: ‘https://github.com/ckyeungac/DeepIRT’.

[48] Zhiwei Wang, Xiaoqin Feng, Jiliang Tang, Gale Yan Huang, Zitao Liu. 2019. ‘Deep Knowledge Tracing with Side Information’. arXiv:1909.00372v1 [cs.A1].